CSCD 212
Final Project Part 1

Important Notes (Read Before Starting the Assignment)

The problem statement below is incomplete. Unlike previous assignments where the problem statement is fully described with a fully commented API, the problem statement given below is incomplete. This is an example of the type of requirements you will encounter in industry.

Do not make assumptions about what is needed! If the information is not provided ask questions.

Your program must be robust. Your program should not assume that only valid inputs are provided. Users can make errors, and the software should catch errors made by users and provide feedback that informs users of the errors and what they need to do to correct the error. We will deliberately test your programs with bad inputs to determine how your program responds.

Problem Description and Constraints

You are required to complete a Java program that implements a basic Airline, (and later Train) Booking System (ATBS). The ATBS allows a client (a user or perhaps another software system) to create airports, airlines, and flights for the airlines.

Each airline is associated with a set of flights. A flight has an originating airport (origin) and a destination airport (destination). The originating and destination airports cannot be the same.

Each flight is associated with a flight section (e.g., first class and business class sections). Each flight section consists of seats organized in rows and columns. The system consists of a SystemManager that provides a single point of access to the functions provided. The SystemManager is a Façade design pattern for the program.

In this assignment, you will be required to implement the following functionality:

PART 1

1. Create an airport. An airport must have a name consisting of exactly three alphabetic characters. No two airports can have the same name. (HINT: Enumerations)
2. Create an airline. An airline has a name that must have a length less than 6. No two airlines can have the same name. (HINT: Enumerations)
3. Create a flight given an airline name, the name of an originating airport, the name of a destination airport, a flight number, and a departure date: A flight has an identifier that is a string of alphanumeric characters.
4. Create a section for a flight. The number of seat rows and columns must be provided when creating a section.
5. Find available flights. Finds all flights from an originating airport to a destination airport with seats that are not booked on a given date.
6. Book a seat. Books an available seat from a given originating airport to a destination airport on a particular date, on a given flight.
7. Print system details. Displays system attribute values for all objects, airports, airplanes, etc.

CSCD 212
Final Project Part 1

Required classes (Design Constraints): Your program must include the following classes.

SystemManager: This class provides the interface (Façade) to the system. That is, clients interact with the system by calling operations in the SystemManager. The SystemManager is linked to all the airport and airline objects in the system. When it is created, the SystemManager has no airport or airline objects linked to it. To create airports and airlines, the createAirport() and createAirline() operations defined in this class must be invoked. The class also contains operations for creating sections of flights (e.g., first class and business class sections), finding available flights between two airports, and booking a seat on a flight. A printout of information on all the airports, airlines, flights, flight sections and seats is obtained by invoking displaySystemDetails().

createAirport(String n): Creates an airport object and links it to the SystemManager. The airport will have a name (code) n; n must have exactly three characters. No two airports can have the same name.

createAirline(String n): Creates an airline object with name n and links it to the SystemManager. An airline has a name that must have a length less than 6. No two airlines can have the same name.

createFlight(String aname, String orig, String dest, int year, int month, int day, String id): Creates a flight for an airline named aname, from an originating airport (orig) to a destination airport (dest) on a particular date. The flight has an identifier (id).

createSection(String air, String flID, int rows, int cols, SeatClass s): Creates a section, of class s, for a flight with identifier flID, associated with an airline, air. The section will contain the input number of rows and columns.

findAvailableFlights(String orig, String dest): Finds all flights from airport orig to airport dest with seats that are not booked.

bookSeat(String air, String fl, SeatClass s, int row, char col): Books seat in given row and column in section s, on flight fl of airline air.

displaySystemDetails(): Displays attribute values for all objects (e.g., airports, airplanes) in system.

Airport: Objects of this class represent airports. The only information maintained is the name, which must be exactly 3 characters in length.

Airline: This class maintains information about airlines. An airline can have 0 or more flights associated with it. When created an airline is not associated with any flights. All flights for a given airline must have unique flight ids.

Flight: This class maintains information about flights. A flight can be associated with 0 or more flight sections. There can only be one flight section of a particular seat class in a flight, e.g., only one business class and only one first class. The seat classes are defined by the enumerator type SeatClass which defines the values, first, business, and economy. The major operations of Flight are summarized below.

CSCD 212
Final Project Part 1

FlightSection: This class maintains information about flight sections. A flight section has a seat class (first, business, or economy) and must have at least 1 seat.

hasAvailableSeats(): returns true iff the section has some seats that are not booked

bookSeat(): books an available seat.

A flight section can contain at most 100 rows of seats and at most 10 columns of seats.

Seat: This class maintains information about seats. Specifically, a seat has an identifier (a seat is identified by a row number and a column character, where the character is a letter from A to J), and a status which indicates whether the seat is booked.

Example Client Class: The following is a sample class with a main() program that calls operations in the SystemManager.

```
public class SampleClient
{
	public static void main(String[] args)
	{
		SystemManager res = new SystemManager();
		res.createAirport("DEN");
		res.createAirport("DFW");
		res.createAirport("LON");
		res.createAirport("DEN");//invalid
		res.createAirport("DENW");//invalid

		res.createAirline("DELTA");
		res.createAirline("AMER");
		res.createAirline("FRONT");
		res.createAirline("FRONTIER"); //invalid
		res.createAirline("FRONT"); //invalid

		res.createFlight("DELTA", "DEN", "LON", 2022, 11, 14, "123");
		res.createFlight("DELTA", "DEN", "DEN", 2023, 1, 18, "567abc");//same airport
		res.createFlight("DEL", "DEN", "LON", 2022, 12, 8, "567"); //invalid airline
		res.createFlight("DELTA", "LON33", "DEN33", 2022, 12, 7, "123");//invalid airports
		res.createFlight("AMER", "DEN", "LON", 2022, 40, 100, "123abc");//invalid date


		res.createSection("DELTA","123", 2, 2, SeatClass.economy);
		res.createSection("DELTA","123", 2, 3, SeatClass.first);
		res.createSection("DELTA","123", 2, 3, SeatClass.first);//Invalid
		res.createSection("SWSERTT","123", 5, 5, SeatClass.economy);//Invalid airline


		res.bookSeat("DELTA", "123", SeatClass.first, 1, 'A');
		res.bookSeat("DELTA", "123", SeatClass.economy, 1, 'A');
		res.bookSeat("DELTA", "123", SeatClass.economy, 1, 'B');
		res.bookSeat("DELTA888", "123", SeatClass.business, 1, 'A'); //Invalid airline
		res.bookSeat("DELTA", "123haha7", SeatClass.business, 1, 'A'); //Invalid flightId
		res.bookSeat("DELTA", "123", SeatClass.economy, 1, 'A'); //already booked

		res.displaySystemDetails();

		res.findAvailableFlights("DEN", "LON");
	}// end main
}// end class
```