Important Notes
(Please read before starting assignment)
Problem statement may be incomplete:
Unlike assignments you may have had previously, in
which the problem is fully described (possibly with bits of code provided for you), the problem statement given below is not necessarily complete or consistent.
This is an example of the type of requirements that you will encounter in industry (actually some real world problem descriptions provide less information than is provided here).

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, when can be done. We will deliberately test your programs with bad inputs to determine how your program responds. You will lose points if your program just dies without any meaningful feedback.

Problem Description and Constraints

Requirements statement

You are required to complete a Java or C# program that implements a basic Airline, (and later Cruise, and Train) Booking System (ACTBS). The ACTBS 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 (i.e. the SystemManager is a Façade for the program).

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

1. Create and airport. An airport must have a name consisting of exactly three alphabetic characters. No two airports can have the same name.
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.
3. Creat 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 attribute values for all objects (e. g., airports, airplanes) in system.

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 invokijng 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 character 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.

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 {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub


		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", 2013, 10, 10, "123");
		res.createFlight("DELTA", "DEN", "DEH", 2013, 8, 8, "567abc");
		res.createFlight("DEL", "DEN", "LON", 2013, 9, 8, "567"); //invalid airline
		res.createFlight("DELTA", "LON33", "DEN33", 2013, 5, 7, "123");//invalid airports
		res.createFlight("AMER", "DEN", "LON", 2010, 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");
        }

}
```

a. Create the initial airport system using information contained in an input file: Information about airports, airlines, flights and their associated information will now be read from a file. When the program first starts it will read the input file and create the initial airport system using the information in the file. The format of the file is given in the section AMS File Format.
b. Store information about the current airport system in a file: The program should be able to store information about the airport system in a file when requested. This feature is similar to the display airport system feature: instead of outputting the information on the screen the information is sent to a file to be stored. The format of this file will be the same as the format of the input airport system file
c. Create flight sections with a layout: In the previous assignment a flight section was created using only the number of seats and columns. In this assignment you will be required to create sections given a layout that identifies window and aisles seats. See the AMS file format section for an example of how the layouts are to be specified.
d. Associate one-way prices for seats on an airline for flights between an origin and destination: Given an airline, the pricing for all seats in a particular flight class for all airline flights between an origin and destination is the same. For example, the price of an economy seat on any American Airways flight from Denver to Seattle is $300, while an economy seat on any USAircorp flight from Denver to Seattle is $200.
e. Book a flight using a seating preference
f. Provide a simple intuitive text-based user interface for the system: The interface will allow a human user to do the following
    a. Create an airport system by using information provided in an input file.
    b. Change the price associated with seats in a flight section (all seats in a flight section have the same price).
    c. Query the system for flights with available seats in a given class (e.g., economy, business) that leave from a specified airport and arrive at specified airport on a particular date. The query operation should list all the available flights found and its prices.
    d. Change the seat class (e.g., economy) pricing for an origin and destination for a given airline.
    e. Book a seat given a specific seat on a flight.
    f. Book a seat on a flight given only a seating preference: The program should allow a user to book a seat on a particular flight using only a seating preference and a flight class. There will only be two seating preferences: Window and Aisle. This booking service will look for an available seat in the flight section with the seating preference. If one is found then the seat is booked. If one is not found, then the system will book any available seat in the specified section, if any.
    g. Display details of the airport system.
    h. Store information about the airport system in a specified file.

AMS File Format AMS information will be stored in a file using the following format:

AMS ::= [list-of-airport-codes] {list-of-airlines}
list-of-airport-codes ::= comma-separated strings
list-of-airlines ::= airline-name1[flightinfo-list1], airline-name2[flightinfo-list2], airlinename3[flightinfo-list3], …

flightinfo-list ::= flightID1|flightdate1|originAirportCode1|destinationAirportCode1[flightsectionlist1], flightID2|flightdate2|originAirportCode2|destinationAirportCode2[flightsection-list2], …
flightdate ::= year, month, day-of-month, hours, minutes
flightsection-list ::= sectionclass: seat-price: layout: number-of-rows, …
sectionclass ::= F, B, E (for first class, business class, economy class)
layout ::= S, M, W (where S is a seat layout with 3 columns with an aisle between columns 1 and 2, M is a seat layout with 4 columns with an aisle between columns 2 and 3, and W is a seat layout with 10 columns with aisles between columns 3 and 4, and between columns 7 and 8)
Example: An airport system with 4 airports, 4 airlines - AMER (in red), UNTD (in green), FRONT, USAIR - and associated flights.
[DEN, NYC, SEA, LAX]{AMER[AA1|2011, 10, 8, 16, 30|DEN|LAX[E:200:S:4,F:500:S:2], AA2|2011, 8, 9, 7, 30|LAX|DEN[E:200:S:5,F:500:S:3], …], UNTD[UA21|2011, 11, 8, 12, 30|NYC|SEA[E:300:S:6, F:800:S:3], UA12|2011, 8, 9, 7, 30|SEA|DEN[B:700:S:5, F:1200:S:2], …], FRONT[…], USAIR[…]}

Extend the program so that it can be used to book cabins on cruise trips as well as seats on flights. The extended program thus supports the building and management of bookings of 2 subsystems: Airline and Cruise subsystems. Write your program to exploit polymorphism and avoid a coding the solution as two completely separated solutions.

Extend your program in the following manner:

1. Develop a password-protected administrator user interface (UI) that allows a system administrator to do the following:
    a. Create airports, airlines, and flights with flight sections and seats.
    b. Create cruises, ports, trips, and ships with cabin sections and cabins.
    c. Print the current state of the airline subsystem including information on seats that are available and booked on each flight.
    d. Print the current state of the cruise subsystem including information on cabins that are available and booked on each cruise trip.
2. Develop a customer UI that allows a customer to:
    a. Find an available seat on a flight and book the available seat.
    b. Find an available cabin on a cruise trip and book the available cabin. Information on cruises A cruise (e.g., the Alaskan Cruise Co.) is associated with a number of trips, where each trip starts on a start-date and ends on an end-date. A trip cannot last more than 21 days. Each trip visits a fixed sequence of ports. Each trip is assigned a ship. Note that a ship can be assigned to more than one trip as long as the trip dates do not overlap. Cabins on a ship are grouped into the following sections: Family (can hold a maximum of 4 persons), Deluxe Family (can hold a maximum of 6 persons), Couples (can hold a maximum of two persons), and Deluxe Couples (can hold a maximum of two persons).

Finally, extend your design to support booking of seats on trains, and you will be required to apply the Abstract Factory design pattern to your design. In the train system, there are different train companies (referred to as trainlines) that manage trains that run between cities. Each trainline has train journeys that stop at cities (i.e., each train journey is associated with a sequence of cities). A train journey has first, business and economy class sections (as in the airline system). Seats in these sections are numbered as in the airline system.