CSCI 1101
Computer Science II

Assignment No. 3
Date Given: Monday, March 6, 2017
Due: Friday, March 17, 2017, 11.55 p.m.

This assignment will test your skills in object-oriented programming with multiple classes and the use of ArrayLists as the data structure. You are to design a program that implements a basic Airline Booking System (ABS). The ABS allows a client program to create airports, airlines, and flights. Each flight has an originating airport (origin) and a destination airport (destination). The origin and destination cannot be the same. Each flight consists of seats organized in rows. Each row has six seats ("A", "B", "C", "D", "E", "F"). Your program should have the following functionality:

1. Create an 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 alphabetic characters. No two airlines can have the same name.
3. Create a flight given an airline name, the name of the originating airport, the name of a destination airport, and a flight number: A flight has an identifier that is a string of alphanumeric characters.
4. Create seats for a flight: The number of rows of seats for this flight is provided.
5. Find available flights: Finds all flights from an originating airport to a destination airport.
6. Book a seat: Books an available seat from a given origin to a given destination on a given flight.
7. Print system details: Displays attributes of all objects (e.g., airports, airlines, etc.) in the system.

Your program must have the following classes:

1. **System Manager class**: This is the main class that aggregates the other classes and provides functionality to the client program. A client program with a main method accesses the System Manager class.

   This class has the following attributes:
   Airports – an arraylist of airport objects, initially empty.
   Airlines – an arraylist of airline objects, initially empty.
   Flights – an arraylist of flight objects, initially empty.

   If required, add other attributes.

   The class has the following methods:

   createAirport(String n): Creates an airport object and updates the appropriate arraylist. The airport will have a name (code) n; n must have exactly three alphabetic characters. No two airports can have the same name.
   createAirline(String n): Creates an airline object with name n and updates the appropriate arraylist. 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, String id): Creates a flight for an airline named aname, from an originating airport (orig) to a destination airport (dest). The flight has an identifier (id).
   createSeats(String air, String flID, int rows): Creates seats with the given number of rows for a flight with identifier flID, associated with an airline, air.
   findAvailableFlights(String orig, String dest): Finds all flights from airport orig to airport dest.
   bookSeat(String air, String fl, int row, char col): Books seat in given row and column on flight fl of airline air, if that particular seat is still available.
   displaySystemDetails(): Displays attribute values for all objects (e.g., airports, flights) in the system.

2. **Airport class**: The only information that is maintained in this class is the name, which must be three alphabetic characters. Add the appropriate get, set and toString methods.
3. **Airline class**: This class maintains information about airlines. All flights for a given airline must have unique ids.
4. **Flight class**: This class maintains information about flights. It has the following attributes:
   a. Airline
   b. Flight id
   c. Originating airport and destination airport
   d. Seats: An array of Seat objects.

Add appropriate attributes (e.g., you could also have the originating and destination airports as attributes for a flight) and methods.

5. **Seat class**: This class maintains information about seats. Each seat has an identifier (a seat is identified by a row number and a column character, which is a letter from A to F). All flights have 6 seats per row. For example, you could have a seat called 2A meaning it is the first seat in the second row, or 10B which means the second seat in row 10). Seat also has a status which indicates if the seat is booked or available.
Add the appropriate attributes and methods.

The following is a sample client program with a main method that calls operations in the SystemManager. You must create your own test program.

```
public class Client{
      public static void main (String[] args){
            SystemManager res = new SystemManager();

            //create airports
            res.createAirport("YHZ");
            res.createAirport("YYZ");
            res.createAirport("YUL");
            res.createAirport("YVR");
            res.createAirport("YYC");
            res.createAirport("LONDON"); //invalid
            res.createAirport("123"); //invalid
            res.createAirport("YEG");
            res.createAirport("BOS");
            res.createAirport("JFK");

            //create airlines
            res.createAirline("AC");
            res.createAirline("DELTA");
            res.createAirline("USAIR");
            res.createAirline("WSJET");
            res.createAirline("FRONTIER"); //invalid

            //create flights
            res.createFlight("AC", "YHZ", "YUL", "123");
            res.createFlight("AC", "YHZ", "YYZ", "567");
            res.createFlight("AC", "YUL", "YHZ", "789");
            res.createFlight("AC", "YUL", "YVR", "123");

            //invalid – AC cannot have two flights with same id.

            res.createFlight("AC", "YHZ", "YYZ", "689");
            res.createFlight("DELTA", "YHZ", "BOS", "123");

            //etc.

            //create seats
            res.createSeats("AC", "123", 40);
            res.createSeats("DELTA", "123", 25);

            //etc.

            //find available flights
            res.findAvailableFlights("YHZ", "YYZ");

            //book seats
            res.bookSeat("AC", "123", 1, 'A');
            res.bookSeat("AC","123",20,'F');
            res.bookSeat("AC", "506", 2, 'B');  //invalid – 506 not created
            res.bookSeat("AC", "123", 55, 'C'); //invalid – row 55 doesn't exist


            //display system details
            res.displaySystemDetails();
```

```
        }
}
```

Submit your all your java files and sample outputs.