CS314 Assignment 1: Object-Oriented Programming Exercise

Due Date: Tuesday, September 11, 2012, before Midnight.

Submit via RamCT

This is a pair-design/programming assignment. You are required to work with a partner on this assignment.

Assignment Objective
The purpose of this assignment is to gauge your current object-oriented programming skills. You will be required to complete a program in which objects interact to accomplish functional goals. In subsequent assignments you will be asked to extend and improve the design you produce in this assignment.

Important Notes (Please read before starting assignment)

Problem statement may be incomplete: Unlike assignments you may have had in previous classes, 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). An important skill that you will start to develop in this course is the skill of analyzing a problem statement to identify missing problem details needed to develop a solution (i.e., a program). In this assignment (and other programming assignments in this course), the first task you should perform is a careful analysis of the problem statement to make sure you have all the information needed to create a solution. Do not make assumptions about what is needed! If the information is not provided, ask questions. There will be a RamCT bulletin board for each assignment where you can post your questions. The TA will try to answer your questions within 24 hours if it has not yet been answered on the bulletin board. You can also ask questions in class if you do not get a response within 24 hours or come in to see either me or the TA during our respective office hours (or at an arranged time)

Your design will be evaluated: Please be aware that in a future assignment another group of students will evaluate and extend your design. You will lose points if the other team finds your program difficult to understand and extend. The design of your program is important; it is not enough to get your program to work; the design should also be consistent with OO principles and be easy to understand by an independent programmer (i.e., a programmer who did not develop the original program).
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 this 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.

Your program must run on department machines: All programs must run on the department systems. If your program does not compile and run on a department machine you will get a 0 for the assignment. You are responsible for ensuring that the program runs on a department machine before the deadline.

Start early: Please start assignments as soon as they are available. The programming assignments in this course are challenging and are not designed to be started and completed a few days before a deadline.

Problem Description and Constraints

Requirements Statement

You are required to complete a Java program that implements a basic Airline Booking System (ABS). The ABS allows a client (a user or 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., SystemManager is a façade for the program – for more information on facades search for the façade design pattern on the web).

In this assignment, you will be required to implement 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. No two airlines can have the same name.

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

### Required Classes (Design Constraints)

You 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 (see bottom of Figure 1). 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, and 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 or not.

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

```java
public class ClientPrrog {
    public static void main(String args[]){
        SystemManager res = new SystemManager();
        //Create airports
        res.createAirport("DEN");
        res.createAirport("DFW");
        res.createAirport("LON");
        res.createAirport("JPN");
        res.createAirport("DE"); //invalid
```
res.createAirport("DEH");
res.createAirport("DEN");
res.createAirport("NCE");
res.createAirport("TRIord9"); //invalid
res.createAirport("DEN");

//Create airlines
res.createAirline("DELTA");
res.createAirline("AMER");
res.createAirline("JET");
res.createAirline("DELTA");
res.createAirline("SWEST");
res.createAirline("AMER");
res.createAirline("FRONT");
res.createAirline("FRONTIER"); //invalid

//Create flights
res.createFlight("DELTA", "DEN", "LON", 2009, 10, 10, "123");
res.createFlight("DELTA", "DEN", "DEH", 2009, 8, 8, "567");
res.createFlight("DELTA", "DEN", "NCE", 2010, 9, 8, "567");
//invalid
res.createFlight("JET", "LON", "DEN", 2009, 5, 7, "123");
res.createFlight("AMER", "DEN", "LON", 2010, 10, 1, "123");
res.createFlight("JET", "DEN", "LON", 2010, 6, 10, "786");
res.createFlight("JET", "DEN", "LON", 2009, 1, 12, "909");

//Create sections
res.createSection("JET","123", 2, 2, SeatClass.economy);
res.createSection("JET","123", 1, 3, SeatClass.economy);
res.createSection("JET","123", 2, 3, SeatClass.first);
res.createSection("DELTA","123", 1, 1, SeatClass.business);
res.createSection("DELTA","123", 1, 2, SeatClass.economy);
res.createSection("SWSERTT","123", 5, 5, SeatClass.economy);
//invalid

res.displaySystemDetails();

res.findAvailableFlights("DEN", "LON");

res.bookSeat("DELTA", "123", SeatClass.business, 1, 'A');
res.bookSeat("DELTA", "123", SeatClass.economy, 1, 'A');
res.bookSeat("DELTA", "123", SeatClass.economy, 1, 'B');
res.bookSeat("DELTA", "123", SeatClass.business, 1, 'A');
//already booked

res.displaySystemDetails();

res.findAvailableFlights("DEN", "LON");

}

}