G51OOP Coursework 3
Aeroplane: Objects and Classes

Question

This coursework requires you to write an object-oriented program to represent a fairly simplistic flight booking system. Please read this specification in conjunction with the sample output document. Please place all your source files in the src directory of your Eclipse project and do not use `packages`.

Each flight is made up of a number of rows of seats in Club Class, and a number of rows of seats in Economy Class.

A row in Club Class consists of 4 seats. The first and last seats on each row are window seats. Every seat in Club Class is an aisle seat; thus some are both.

A row in Economy Class consists of 6 seats. The first and last seats on each row are window seats. The middle two seats are aisle seats. The remaining two seats are neither window nor aisle seats.

The following shows this in pictorial format:

(W = window, A = aisle, B = both, N = neither)

Club: B A A B

Economy: WNA ANW

The number of rows in each seating class is to be specified by the user.

When the program is run, the user should be presented with the following menu:

```
Flight Booking Menu
===================
1) Create Flight
Q) Quit

Enter your choice:
```

If the user enters 'q' or 'Q', the program should print "`Quitting....`" and then exit.

If the user enters '1', the Flight Creator menu should be shown, and the user should be prompted for the destination and departure time. (See the sample output below for full details of the expected output.)

The user should then be prompted for the number of rows in each seating class (again, see the sample output). You should then return to the main menu. Once at least one flight has been added, the menu should look the same as:

```
Flight Booking Menu
===================
1) Create Flight
2) Make Booking on Flight
3) Print A Flight's Manifest
Q) Quit

Enter your choice:
```

If the user enters '2' or '3', a list of all available flights should be printed to the screen. The user should then be prompted to select one of these flights by its flight number -- this should simply be the order in which the flights were created, starting with zero. If the user entered '3', then once they have selected a flight, you should print the flight's manifest in the following format:

```
Flight Manifest
Destination: Destination Name Here
Departure time: Departure Time Here
Club Class passenger list:
Row: 0, Column: 2, Name: Passenger1, Nationality: Nationality1
Row: 1, Column: 3, Name: Passenger2, Nationality: Nationality2
Economy Class passenger list:
Row: 0, Column: 0, Name: PassengerX, Nationality: NationalityY
Row: 0, Column: 3, Name: SomePassengerName, Nationality: SomeNationality
```

If there are no passengers in a seating class, print the message

```
No passengers in Club Class
```

or

```
No passengers in Economy Class
```

as appropriate, instead of the passenger list.

If the user entered '2', once they have selected a flight, they should be prompted for a seating class (club or economy). (See sample output.) If all seats in one of the classes are full, do not print the corresponding menu option.

Once they have selected a seat type, they should be presented with the Seat Selection menu:

```
Seat Selection
==============
1) Pick Specific Seat
2) Any Window Seat
3) Any Aisle Seat
4) Any Seat
```

```
Enter Preference:
```

If the user enters '1', a seating plan should be printed to the screen (see sample output and further description below) and the user prompted for a row and column number.

If the user enters '2', the first available window seat should be automatically selected.

If the user enters '3', the first available aisle seat should be automatically selected.

If the user enters anything else, the first available seat of any type should be selected.

For all of the above options, the user should then be prompted for the Person's name and nationality, and then asked if they wish to book another seat on the flight. If they enter 'y' or 'Y', they should be prompted with the Seat Type menu again (see sample output). If they enter anything else, return to the main menu.

You are provided with the following files:

FlightBookingSystem.java

Contains the main method, and should contain the majority (if not all) of the code for the various menus. It must also store a list of Flights; we suggest you use an array of Flight Objects.

Constructor:

You should declare one constructor for the FlightBookingSystem class. It should take one integer as an argument, used as the maximum number of storable flights. Hint: use this to set the size of the array of Flight Objects.

Required methods:

main

This is provided for you. You should not alter it. It creates a new FlightBookingSystem object with space for 10 flights, and then calls the runMainMenu method on it.

runMainMenu

This method should be declared public, and should take no parameters and return nothing. This should contain the main logic behind the menu system, though you may use helper methods internally as appropriate. Any helper methods you create must be declared private.

IFlight.java

The Java interface for the Flight class. You must not alter this interface. The Flight.java class must `extend` this interface.

Flight.java

Represents a flight. It must store a destination and a departure time, and a list of seats. It should implement all the methods in the IFlight.java interface and any others necessary.

Constructor:

The Flight class should have one constructor. It should take two ints and two Strings, representing the number of rows in Club Class, the number of rows in Economy Class, the destination, and the departure time. (You may read and store the departure time as a String, and accept any value.)

Required methods (see IFLight.java):

getDestination
Takes no arguments, and returns a String, the destination.

getTime
Takes no arguments, and returns a String, the departure time.

numRows
Takes a boolean (true for Club Class, false for Economy Class) and returns an int, representing the number of rows in that class.

numCols
Takes a boolean (true for Club Class, false for Economy Class) and returns an int, representing the number of columns in that class.

addSeatBooking
Takes a Location Object and a Person Object as arguments, and returns nothing. Sets the Seat at the specified Location to hold the specified Person Object.

isSeatEmpty
Takes a Location Object, and returns a boolean. Returns true if the seat at the specified Location has no Person assigned.

getNextAvailableAisleSeat
Takes a boolean (true for Club Class, false for Economy Class) and returns a Location Object, representing the next available aisle seat in the specified seating class. If there are no window seats available, the method should return null instead of a Location Object. The 'next available seat' is defined as the topmost, leftmost seat which has no Person assigned to it and fulfils the requirement of being an aisle seat.

getNextAvailableWindowSeat

Identical to getNextAvailableAisleSeat, except that it should return the first available window seat.

getNextAvailableSeat

Identical to the above, except that it should return the first available seat of any type.

printManifest

Takes no arguments and returns nothing. When called, this method should print out the flight's manifest in the following format:

```
Flight Manifest
Destination: Destination Name Here
Departure time: Departure Time Here
Club Class passenger list:
Row: 0, Column: 2, Name: Passenger1, Nationality: Nationality1
Row: 1, Column: 3, Name: Passenger2, Nationality: Nationality2
Economy Class passenger list:
Row: 0, Column: 0, Name: PassengerX, Nationality: NationalityY
Row: 0, Column: 3, Name: SomePassengerName, Nationality: SomeNationality
```

If there are no passengers in a seating class, print the message

```
No passengers in Club Class
```

or

```
No passengers in Economy Class
```

as appropriate, instead of the passenger list.

printSeatingPlan

Takes one argument, a boolean, and returns nothing. The Boolean is used to determine whether to print the seating plan of Club Class or Economy Class.

For Club Class with 2 rows, this should be printed in the following format:

```
  0 1 2 3
0:X O O X
1:X O O O
```

where O indicates an empty seat, and X indicates a full one. Economy Class should look similar, except with columns numbered 0 to 5.

Seat.java

Represents a seat on a flight. Each seat must store whether it is a window seat, and whether it's an aisle seat. (Note that since some seats in Economy Class are neither, and some seats in Club Class are both, one boolean will probably be insufficent.) Each seat must also be able to have one person assigned to it, though it may also be empty.

Constructor:

There should be one constructor, which takes two booleans as parameters. One should specify whether the seat is a window seat, and the other should specify whether it's an aisle seat.

Required methods:

getPerson

This should take no arguments, and return the Person Object stored within this Seat Object. (If there is no person assigned to the seat, it is OK to return null. Just make sure you check for it whenever you call getPerson().)

setPerson

This should take one argument, a Person Object, and return nothing.

Calling this method should have the effect of assigning the

specified Person to this Seat.

isWindow

Takes no arguments, and returns a boolean depending on whether this

is a window seat.

isAisle

Takes no arguments, and returns a boolean depending on whether this

is an aisle seat.

Person.java

Represents a person. A Person has a name and a nationality.

Constructor:

There should be one constructor. It should take two Strings as arguments:

the person's name, and the person's nationality.

Required methods:

getName

Takes no arguments, and returns a String, the Person's name.

getNationality

Takes no arguments, and returns a String, the Person's nationality.

Location.java

This class is already implemented for you, and you should not change it. You should use a Location object wherever you find you need to return a seat's location, for example in the format

```
return new Location(0, 1, true);
```

This would have the effect of specifying seat 0,1 in the Club Class.

Note: The automated marking will only mark files it knows about. If you create other

.java files, they will not be marked, and your code will probably not work.

Error Conditions

At the Flight Booking Menu, if the user enters an invalid menu option, print

```
ERROR: Invalid menu choice!
```

At the Aeroplane Details menu, if the user enters a number of rows less than zero, print the message

```
ERROR: Invalid Row Count
```

At the Available flights menu (shown when making a booking, or printing a manifest) if the user enters an invalid flight number, print the message

```
ERROR: Invalid menu choice!
```

At the Seat Type menu, if the user enters anything other than 'c' or 'C', treat it as if they intended to press 'e', ie you should NEVER print an error message from this menu.

At the Seat Selection menu, if the user does not make a valid menu choice, treat it as if they had intended to press '4', ie you should NEVER print an error message from this menu.

When asking the user if they wish to book another seat, accept 'y' or 'Y' as yes, and any other character as no. You should not print an error message here.

When booking a specific seat, if the requested seat is already booked, print the Message:

```
ERROR: Seat already taken
```

and prompt for another row and column.

When booking a specific seat, if the provided row or column number is invalid (eg if is less than zero or greater than the number of rows/columns) print the message:

```
ERROR: Invalid row number
```

or

```
ERROR: Invalid column number
```

as appropriate, and then prompt for another row and column.

When booking any window seat, if there are no window seats available in the specific seating class, print the message

```
Sorry, no window seats left
```

Similarly, if there are no aisle seats left in the specific seating class when booking an aisle seat, print the message:

```
Sorry, no aisle seats left
```

In both of the above cases, prompt the user if they would like to book another seat, as if they had successfully made a booking.

If the user tries to make a booking on a flight that is fully booked, print the Message:

```
Flight is fully booked, Sorry
```

If the user tries add a flight that the flight array is full, print the Message:

```
Total Flights available exceeded
```

Submission

Submission is via Moodle. You should submit a .zip file containing only the required .java files and G51OOPInput.java. You should not submit any other format other than .zip (no .iso, .rar, .7z or other alternatives). Make sure to submit the .java files, and not the .class files. You must not use packages in your submission.