Assignment 2: Airline Reservation App

Due date: Friday, 30th September 2016 at **12noon** on BlackBoard.
This assignment is worth 10% of your final grade and has 100 marks in total.

Brief

For this assignment, you will individually develop an Airline Reservation Application in Java enabling airlines to view the seating map of a flight and to make seat reservations in first or economy class, based on their reservation policies. For a given flight, the Airline Reservation Application retrieves its seating map partitioned into first and economy classes as depicted in the left hand side of Figure 1.

A seat request marks a seat as reserved. For example, the first class seat **3E** is reserved and shown in the left hand side of Figure 1 as a solid circle. This seat is an *aisle* type of seat, since it is next to an aisle. Similarly, *window* type seats are next to the aircraft's windows (such as **5A**), and all other seats are *middle* type seats (such as **9D**).

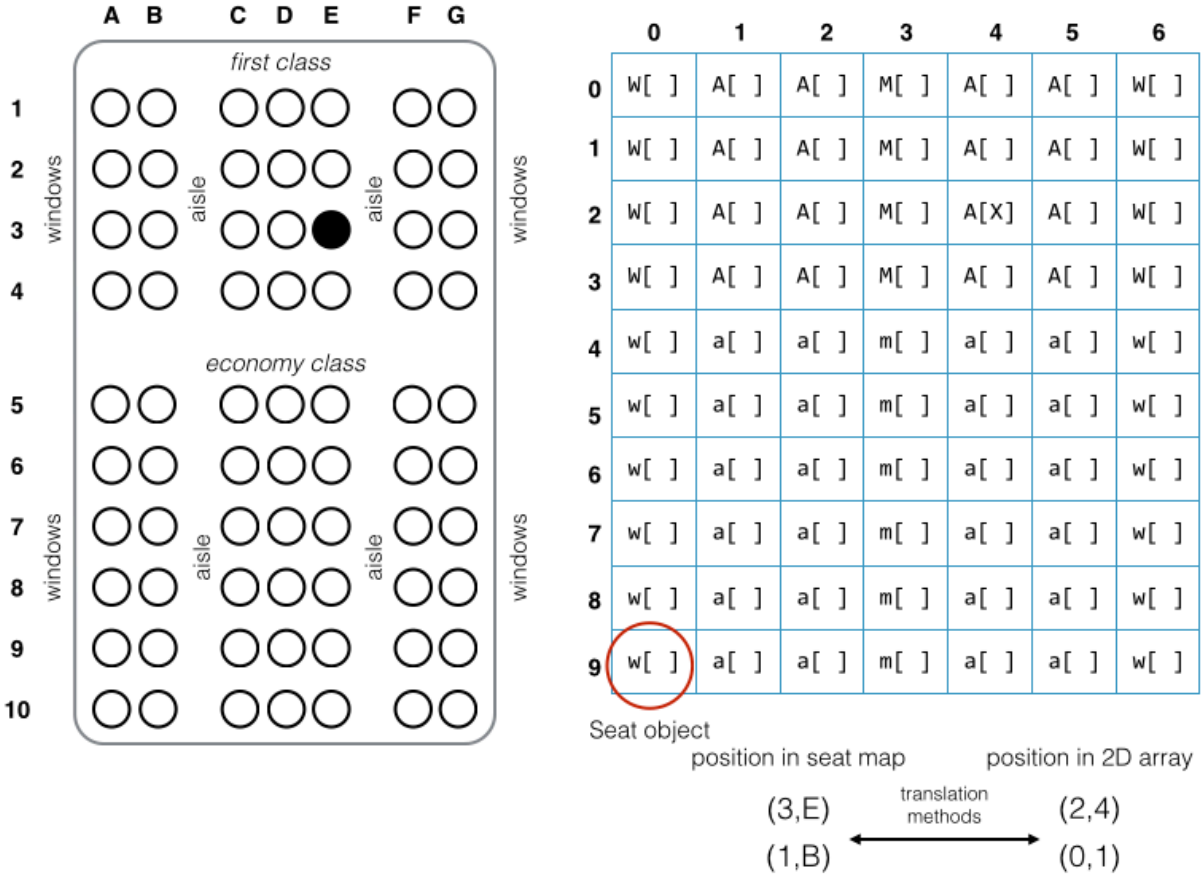

Figure 1: Seat map stored as a 2D array of Seat objects

Airline Reservation Policies

Different airlines have different reservation policies if the requested seating is unavailable on a flight. Consider the policies of the following two airlines:

SimpleJet

1. **Seating Reservations in First Class**
   - Find and reserve a seat in first class that matches the requested seat type
   - If no such first class seat with the matching type exists, then find and reserve any seat in first class
   - If there are no seats available in first class then find and reserve a middle or window seat in economy class, also reserving one of the neighboring seats (for extra passenger room)
   - If there are no seats matching any of these criteria then a reservation cannot be made
2. **Seating Reservations in Economy Class**
   - Find and reserve a seat in economy class that matches the requested seat type
   - If no such economy class seat with the matching type exists, then find and reserve any seat in economy class
   - If there are no seats matching any of these criteria then a reservation cannot be made

RynoAir

1. **Seating Reservations in First Class**
   - Find and reserve a seat in first class that matches the requested seat type
   - If no such first class seat with the matching type exists, then find and reserve any seat in first class
   - If there are no seats available in first class then find and reserve a middle seat in economy class, also reserving both the left and right neighboring seats (for extra passenger room)
   - If there are no seats matching any of these criteria then a reservation cannot be made
2. **Seating Reservations in Economy Class**
   - Find and reserve a seat in economy class that matches the requested seat type
   - If no such economy class seat with the matching type exists, then find and reserve any seat in economy class
   - If there are no seats matching any of these criteria then a reservation cannot be made

Methodology

You will develop a number of classes in Java that implement the Airline Reservation Application. For full marks, your classes **must** adhere to correct OOP design principles.

Flights

Design a class to store information relating to a **Flight**. For example, start and destination cities, departure time, flight number and a **SeatMap** object that stores the seat map (see below). Consider the data you will need to store to implement the functionality of the Airline Reservation Application.

Storing Seat Maps and Reservations

In this section you will develop classes to store seat reservations.

Design a **Seat** class which has data to store whether or not a seat is reserved or is in first class. The seat has data that stores the type of the seat: AISLE, MIDDLE, WINDOW (Hint: consider using an enumerated type). The **Seat** stores a **SeatPosition** object which contains the row and column of a seat in the seat map: e.g. **1A**, **6C**, **2G**, forming a seat map position. Write a toString method which returns a **Seat** representation according to the right hand side of Figure 1. Write another method that outputs a longer text description of the **Seat**. For example: `Economy class MIDDLE seat at: 9D is not reserved.`

Develop an *abstract* **SeatMap** class with an abstract method void initialiseSeatMap(). The class stores a two-dimensional array of **Seat** objects according to the right hand side of Figure 1, to be initialized by concrete classes (see below). **SeatMap** contains instance variables that store the number of rows and columns that comprise the matrix and maintains the number of rows situated in first class.

All instance variables are **protected** in the **SeatMap** class. You should not write any set methods. Instead, it has the following functionality to query the seat map:

1. Provide some accessor methods that return the last row (e.g. 10) and last column (e.g. the character 'G') in the seat mapping.
2. Seat getSeat(int, char) returns the seat in the specified seat map position (Refer to Figure 1).
3. getLeft (Seat)/getRight(Seat), returns a seat to the left or right of the input Seat. (Hint: use SeatPosition to find the seat in the 2D array). If the seat does not exist, return null.
4. Seat queryAvailableEconomySeat(SeatType) returns a seat in economy that has the matching SeatType and is not already reserved. If all these types of seats are reserved, return any seat in economy. If all seats are reserved, return null.
5. Seat queryAvailableFirstClassSeat(SeatType) is the same, but only searches first class.
6. toString returns a string containing a text representation of the **SeatMap** similar to the right hand side of Figure 1.

Develop two classes that extend **SeatMap**:

1. **BoeingSeatMap** is a concrete class which has a default constructor initialising the number of rows to 10, the number of columns to 7 and the number of first class rows to 4. The constructor initialises the **seats** array accordingly. The constructor calls the **initialiseSeatMap** method. Write an algorithm in **initialiseSeatMap** to populate the **seats** array according to Figure 2.
2. **AirBusSeatMap** is a concrete class, which has a default constructor initialising the number of rows to 12, the number of columns to 9 and the number of first class rows to 6. The constructor initialises the **seats** array accordingly. The constructor calls the **initialiseSeatMap** method. Write an algorithm in **initialiseSeatMap** to populate the **seats** array according to Figure 2.

Note: **initialiseSeatMap** instantiates each **Seat** object stored in the seats array with the appropriate SeatPosition object by translating the array indices to a SeatPosition.

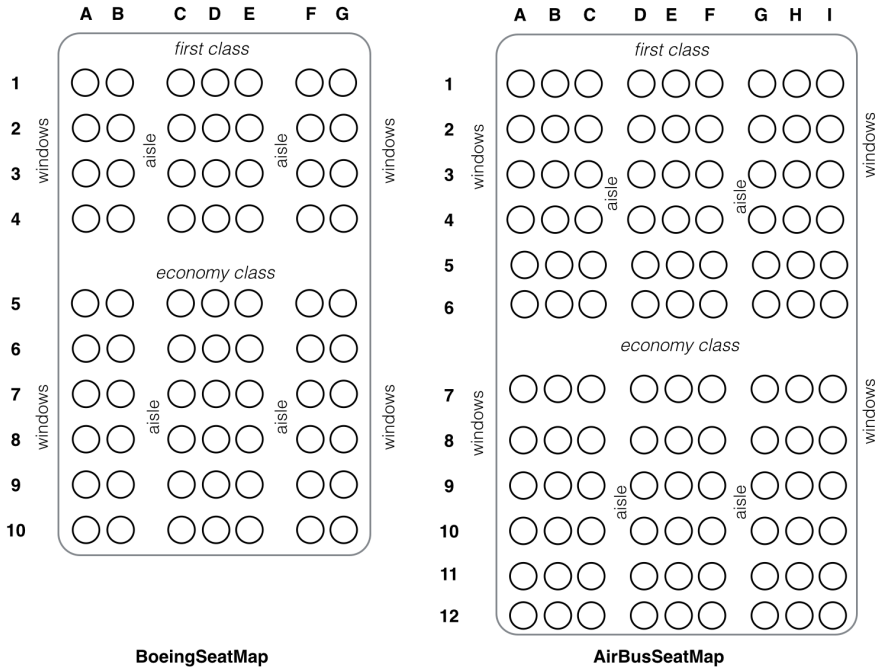

Figure 2: SeatMap for Boeing and AirBus

Making Airline Reservations

Airlines query a flight's seating map to determine the availability of seats. Create an abstract class **Airline** that stores the name of the airline, with useful accessor and mutator methods. Define a toString method that prints out a nice representation of the object. This class has two *abstract* methods: Seat reserveFirstClass(Flight, SeatType) and Seat reserveEconomy(Flight, SeatType)

Airline is extended by two concrete classes **SimpleJet** and **RynoAir** and each class implements the abstract methods *according to the airline's reservation policies.*

Program Interaction

Develop an application class called **AirlineReservationApplication** which interacts with the core functionality of the Airline Reservation Application. Your program should involve multiple flights to select from, different seating configurations and demonstrate both airline reservation policies. You may consider writing additional methods to simulate when economy or first class is fully reserved.

Marking Scheme

Criteria:	Weight:	Grade A Grade Range: 100 ≥ x ≥ 80%	Grade B Grade Range: 80 > x ≥ 65%	Grade C Grade Range: 65 > x ≥ 50%	Grade D Grade Range: 50 > x ≥ 0%
Functionality of seat map classes	**35%**	OOP paradigm consistently used for implementation of all seat map functionality.	Inconsistent use of OOP paradigm but correct implementation of seat map functionality	Incorrect seat map functionality and poor use of OOP paradigm	Absent seat map functionality or code does not compile
Functionality of airline classes	**20%**	OOP paradigm consistently used for implementation of all airline functionality.	Inconsistent use of OOP paradigm but correct implementation of airline functionality	Some basic functionality of airline classes/poor usage of abstract and concrete classes	Absent functionality of airline classes or code does not compile.
Program's Runtime Demonstration: -Uniqueness -Purpose -Context -Interactive	**15%**	The object-oriented program is unique, purposeful and provides an elegant implementation of the Airline Reservation Application. Program is interactive.	The object-oriented program is unique, purposeful. Reasonable implementation of the Airline Reservation Application. Program is interactive.	The object-oriented program features an incomplete demonstration of the Airline Reservation Application Program is not interactive.	Absent functionality of Airline Reservation Application or code does not run after compiling
Code Quality: -Whitespace -Naming -Reuse -Modularity -Encapsulation	**15%**	Whitespace is comprehensively consistent. All naming is sensible and meaningful. Code reuse is present. Code is modular. Code is well encapsulated.	Whitespace is comprehensively consistent. Majority of naming is sensible. Code is modular. Code is encapsulated.	Whitespace is comprehensively consistent. Code has some modularity. Code has some encapsulation.	Whitespace is inconsistent and hence code is difficult to read.
Documentation Standards: -Algorithms Commented -Javadoc	**15%**	Entire codebase has comprehensive Javadoc commenting. Algorithms are well commented.	Majority of the codebase features Javadoc commenting. Majority of algorithms are commented.	Some Javadoc comments present. Some algorithms are commented.	No Javadoc comments present.

Javadoc Commenting

1. Your classes must have commenting of the form:

```
/**
  * Comment describing the class.
  * @author kjohnson studentnumber
 **/
```

2. All methods must be commented with appropriate Javadocs metatags. For example:

```
/**
 * A comment to describe the method
 * @param a parameter description
 * @return a description of the returned result
 * @author kjohnson studentnumber
 * */
```

Submission Instructions

1. Download and import the Java project **Assignment2** to your Eclipse workspace
2. Develop and store all your source code in the **reservations** package
3. Save several example runs of your airline reservation as text files in the **output** directory located in this package. These should have the naming format: studentID-sampleRun1.txt, studentID-sampleRun2.txt, etc, with your student ID number
4. Use the export feature in Eclipse to export your **Assignment2** resources to an archive file with the naming format: Lastname-firstName-studentID.zip.
5. Upload your archive file to the Blackboard sys

Late submissions will receive a grade of 0

An extension will only be considered with a Special Consideration Form approved by the School Registrar. These forms are available at the School of Engineering, Computer and Mathematical Science located in the WT Level 1 Foyer.

You will receive your marked assignment via the Blackboard Grade Centre. Please look over your entire assignment to make sure that it has been marked correctly. If you have any concerns, you must raise them with the lecturer. You have **one week** to raise any concerns regarding your mark. After that time, your mark cannot be changed.
Do not email the lecturer because you do not like your mark. Only go if you feel something has been marked incorrectly.

Authenticity

All work submitted must be unique and your own -
We use automated methods to detect academic integrity breaches.