1 Object Oriented Programming Assignment

Airports are managed by a myriad of different information systems. In this assignment we focus on one such system. You are required to complete a Java program that implements a basic Airport Gate Management System (GMS). The system is described in Section 2. It allows you to build a network of terminals and gates that incoming planes are assigned to. We do not concern ourselves directly with the departure process. Although superficially object oriented, it does not sufficiently encapsulate the state of an airport. The main learning outcome of this assignment is to improve and justify your version of the GMS.

We suggest you implement the initial version described in Section 3 and then refactor it. However you are not obliged to do this.

You are expected to submit a document containing a brief introduction stating what you have achieved, a critique of the initial GMS and an annotated design of the improved version. Include code fragments and sample runs if you feel that they supplement your narrative. Your main document *should not exceed ten pages*. You should attach two appendices to your document, both using a small font. The first should contain your GMS and the second should contain your manual testing to highlight that your system performs correctly on both good and bad inputs. The use case given in this brief does not explore the boundary conditions so you will have to add more incoming flights to reveal the problematic scenarios.

We are looking for your ability to apply object oriented concepts through the Java language and your understanding of the pros and cons of various design decisions.

2 Requirements Statement

The GMS allows a client (a user or another software system) to create an *airport* with *runways*, *terminals* and *flights*. Its main task is to manage runway selection and gate assignments.

An airport has a number of runways, with potentially differing lengths. Large planes need long runways, midsize planes can land on either medium sized runways or longer. Small planes can land on short runways or longer ones too.

An airport has a number of terminals, each terminal has a number of gates. Every day there will be a list of planes that will be landing at the airport. You will need to schedule the runways and gates for each plane.

The system consists of a `GateManager` that provides a single point of access to the functions provided (i.e., `GateManager` is a façade for the program – more information can be found by searching for the façade design pattern on the web).

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

- Create an airport and give it a name.
- Create a number of distinct runways. Each can be either `short`, `medium` or `long`.
- Create a number of different airlines, each will have their own name.
- Create an arrival flight given an airline name, a numeric id, an arrival time, minimum size of runway needed, and a departure time.
- Develop a schedule that maps flights to runways and subsequently gates.
- Display the schedule.

3 Required Classes for the Initial Version of the GMS

The basic version of the GMS includes the following classes.

3.1 GateManager

This class provides the interface (façade) to the system. That is, clients interact with the system by calling operations in the `GateManager`. The `GateManager` is linked to all the airport objects in the system. When it is created, the `GateManager` has no terminal, runway or airline objects linked to it. To create such entities, the `createTerminal()`, `createRunway()` and `createAirline()` operations defined in this class must be invoked. The class also contains operations to generate a schedule mapping incoming flights to runways and gates.

- `createAirport(String n)`: Creates an airport object and links it to the GateManager. No two airports can have the same name.
- `createTerminal(String n, int number)`: Creates a gate object and links it to the GateManager. A gate has a name and a number of gates. No two gates can have the same name.
- `createAirline(String n)`: Creates an airline object with name `n` and links it to the GateManager. No two airlines can have the same name.
- `createFlight(String anm, int id, String atm, Runway r, String dtm)`: Creates a flight landing request for an airline named `anm`, with a unique `id` number, and an arrival time `atm` given in 24 hour format, the size of runway `r` required, and a departure time `dtm` also given in 24 hour format.
- `calculateFlightSchedule()`: Attempts to match incoming flights with runways and terminals. It will report an error if not able to do so.
- `displayFlightSchedule()`: Displays flight landing details; including runway and gate allocation.

3.2 Terminal

This class maintains information about terminals. When created a terminal will have a name and one or more terminals associated with it.

3.3 Runway

This class maintains information about runways. When created a runway will have a name and a runway length; either `short`, `medium` or `long`. No two runways can have the same name.

3.4 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.

3.5 Flight

This class maintains information about incoming flights. All flights for a given airline must have unique flight identity numbers. They will have an expected arrival time and stipulate the minimum sized runway required in order to be able to land and take off. Flights will also know their departure time, small airplanes can be turned around in thirty minutes, medium sized one's require an hour. Large planes require ninety minutes to ensure passengers can disembark, the plane cleaned and the next set of passengers to be seated. Flights must have left their gate before they can be expected to return to the airport.

4 Example Client Class

The following is a sample class with a `main()` program that calls operations in the `GateManager`.

```
public class ClientProg {
public  static void main(String args[]){
     GateManager gm = new GateManager("Oxford Airport");

     //Create terminals with number of gates
     gm.createTerminal ("T1", 4);
     gm.createTerminal ("T2", 8);

     //Create runways, with differing lengths
     gm.createRunway ("North",Runway.long);
     gm.createRunway ("South",Runway.medium);

     //Create airlines
```

```
        gm.createAirline("OXAIR");
        gm.createAirline("BRITISH");
        gm.createAirline("FLYCAM");
        gm.createAirline("AIRFRANCE");
        gm.createAirline("AERLINGUS");

        //Create flights
        gm.createFlight("OXAIR", 1, "10:00", Runway.short, "11:00");
        gm.createFlight("RYANAIR", 666, "10:05", Runway.medium, "10:30");
                    // invalid, undefined airline
        gm.createFlight("AERLINGUS", 45, "10:10", Runway.medium, "16:10");
        gm.createFlight("OXAIR", 1, "10:30", Runway.short, "11:30");
                    // invalid, plane will have already arrived
        gm.createFlight("FLYCAM", 1, "11:00", Runway.short, "12:20");
        gm.createFlight("BRITISH", 45, "11:30", Runway.long, "12:00");
                    // invalid, a large plane needs more time to turnaround
        gm.createFlight("AIRFRANCE", 909, "12:00", Runway.long, "19:00");
        gm.createFlight("OXAIR", 2, "14:00", Runway.short, "15:00");
        gm.createFlight("FLYCAM", 6, "15:00", Runway.short, "16:20");

        gm.createSchedule();
        gm.displaySchedule();
} }
```

5 Object Flight based GMS

The manner in which gate bookings are modelled in the Basic GMS is rather imperative. A more sensible object based approach should be developed.

Task 1: Critique the current implementation from a Software Engineering perspective. Design a more Object Oriented version, justifying your decisions, and implement it.

6 Optional: Responding to High Demand

In the current implementation we can only book a runway and gate if there is availability. In practice flights can circulate for up to half an hour before needing to land. Similarly flights can wait to be docked at a gate if there isn't one available.

Optional Task 2: We want to provide the facility to join a holding pattern so that same arrival time planes can circulate if need be. This can be modelled quite simply by extending the arrival time. A gap of five minutes is required

between landings. Flights can wait on the tarmac for up to fifteen minutes to access a gate.