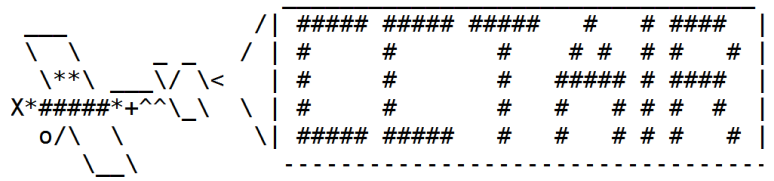

COLLEGE OF COMPUTING TECHNOLOGY - DUBLIN
BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY

OBJECT ORIENTED CONSTRUCTS

Assessment 1 – Design of an airline schedule

Miguelantonio Guerra - 2016324
Asmer Bracho - 2016328
Adelo Vieira - 2017279

Lecturer: Mr. Mark Morrissey

April 1, 2018

Contents

List of Figures

1 Project description

The aim of this assignment is to develop an Java application for an airline schedule.

The program must includes:

- A class called **AirPlane** that will store data about a commercial aircraft
  - Data for an airplane includes (but not limited to):
    - Make (type of plane, such as, “Boeing”)
    - Model (model number, for example, 707)
    - Capacity (seating capacity of the plane - a whole number up to 400)
    - Pilot (name of pilot assigned to the aircraft)
  - The class also must include:
    - A **constructor** that accepts parameters in the order given above and initializes the corresponding instance variables
    - Getter methods as given below:
      - **getMake()** - return a string with the make of the airplane
      - **getModel()** - return a number being the airplane’s model
      - **getPilot()** - return (at a minimum) the pilot’s name
      - **capacity()** - return a number being the capacity of the airplane.
    - A method called **assignPilot** which accepts the name of a pilot and assigns the pilot to the airplane (this may also be created as an object to include the rating of the Pilot, ie. What type of planes they are qualified to fly).
    - A **toString()** method which should return a string with the format:
      Airplane Information:
      Aircraft : «insert make» «insert model»
      Capacity: _____ seats
      Pilot: _____

- A class called **Flight** that will store data about a commercial flight.
  - Data includes:
    - Origin (e.g. “Dublin”)
    - Destination (e.g. “New York”)
    - Departure time (e.g. “10:10”)

    - Arrival time (e.g “13:30”)
    - Date of flight (e.g. “05/12/2015”)
    - Aircraft assigned (this should be an object)
  - The implementation should include:
    - A **constructor** which accepts data (except for departure time and arrival time), in the order listed above and initializes each relevant instance variable accordingly.
    - A **toString()** method which should return a string with the format:
      Flight Information:
      Date: «insert date of flight»
      From: «insert origin» to «insert destination»
      Flight time: «insert departure time» to «insert arrival time»
      Plane Information:
      Aircraft : «insert make» «insert model»
      Capacity: _____ seats
      Pilot: _____
    - Also define the following methods(overload) that will set the scheduled time for a flight:
      - void schedule(String arrivalTime)
      - void schedule(String arrivalTime, String departureTime)
- Implement a driver class (main method), called **CCTAir** that will do the following:
  - Declare and initialize a number of flights
  - Add up to 5 flights of your choice.
  - Use the second version of the schedule method to set the time schedule for a flight
  - Use the first version of the schedule method to update the arrival time for a flight in the list
  - Display all flights

2 Requirements gathering and analysis

The system analysis was made from:

- The requirements explained in the project description (Section 1).
- A scenario in which the function of the system (step by step with a typical user) is detailed.

We first determine the main **Use case** of the system:

- Display Flights
- Update Flight
- Display Airplanes
- Display Pilots

In order to build the system requirements we visualized a scenario in which a typical user uses the system (Table 1)

Description			**Implications** A more detailed reasoning of the classes is shown in the section 3.
After the user enters the system, it will show: - **A welcome message** following by - **A Main Menu** showing the Main Menu of the System: Flight, Airplanes, Pilots			A class **Menu** We will create a method for each submenu: **MainMenu()** **MunuFlight()** **MenuAirplanes()** **MenuPilots()** The Menu will also have the option of **Exit** and **go back** to the previous Menu. So we will need these methods: **goBack()** We will also implement a method in charge of cleaning the console when changing the menu: **deleteMenu()**
	Flight:	This option will allow entering the Flight Submenu	Class **Flight** A method **MunuFlight()** will show the options available for Flights
	Display all Flights	This options will show a list of all Flights created in the system	A Method **showFlightsSimplified()** will be required
	Update Pilot	This options will allow to select from the list of pilots	**updateFlightsPilot()**
	Update Copilot	Similar to Update Pilot	
	Update Schedule	This option will allow the user to enter a new Schedule for a selected flight.	A method **updateSchedule()** will be implemented. Since the user will be asked to enter the new time, a method should be implemented in order to validate the time has be entered has the correct format.
	Update Arrival time	Simiar to Update Schedule but just for arrival time	**updateArrival()**
	Airplanes	When this option is entered, the system should displays all the Airplanes registered	**Class AirPlain** **MenuAirplanes()**
	Pilots	Display all Pilots working at AirCCT	**Class Staff** **Class Pilot**

Table 1: Functions of the program

3 Classes diagram and reasoning behind the choices

Some importan consideration about the Class design:

- ***Class Data:*** It is important to notice that in the first version of our system (Test Version 1.0) the data will be created on execution and the system won't be able to storage the data on a Database or in some other way. That is why we decided to create a class responsible to create the Data will be used (class Data)
- ***Class Pilot:*** (subclass of the class *Staff*)
  - It is a logical assumption to say that in a Airline Pilots will be part of a Staff. So we will create a class Staff (superclass) and the Pilot class (subclass) will inherit the properties of Staff.
  - In addition to the Staff parameters (staffId, name) the Pilot class will have the parameter «licese». This parameter will set the rating of the Pilot (What type of planes they are qualified to fly):
    * License A: Senior Pilot allow to fly any Airplane.
    * License B: Allow to fly Airbus and Bombardier
    * License C: Allow to fly only Bombardier
- ***Class AirPlain:***
  - In addition to the Pilot parameter (pilot assigned to the airplan) that was required on the description of the project. We will also assign a coPilot to the Airplan.
  - It is important to notice that a coPilot is represented by the same object Pilot, since a coPilot is a staff member (Pilot) acting in this case as a co pilot.
  - It will also be necessary to include the parameter «licenseRequired», which will define the license (A, B or C) that is required to be the pilot of the plane. No specific license will be required to act as copilot.
  - A first constructor that takes parameter including only one pilot will be implemented. this constructor will be use for planes which capacity does not exceed 200 passengers
  - A second constructor will be implemented to create an Airplane with a pilot and copilot. This constructor will be use for planes which capacity is bigger than 200 passengers
- ***Class Menu:*** We decided the application will be manage from a selection Menu in the NetBeans Console. That means, a Menu will be displayed in the console showing available options. The user will enter the desired option indicating a number associated with the option.

We decided to organize the classes in different package in the following way:

- **cctair:**
  - CCTAir (Main class)
  - Data
- **cctair.staff:**

  - Staff (superclass)
  - Pilot (subclass)
- **cctair.plane:**
  - InterfaceAirPlane
  - AirPlane
- **cctair.flight:**
  - InterfaceFlight
  - Flight
- **cctair.menu:**
  - InterfaceMenu
  - Menu

In Figure 3.1 we show the Class diagram we obtain at the end of the project. From this diagram it is pertinent to notice the relationship between the different classes. In Section 4 we discuss the functions of Interfaces created. Because of the simplicity of the class Staff, we didn't crate an interface for this class.

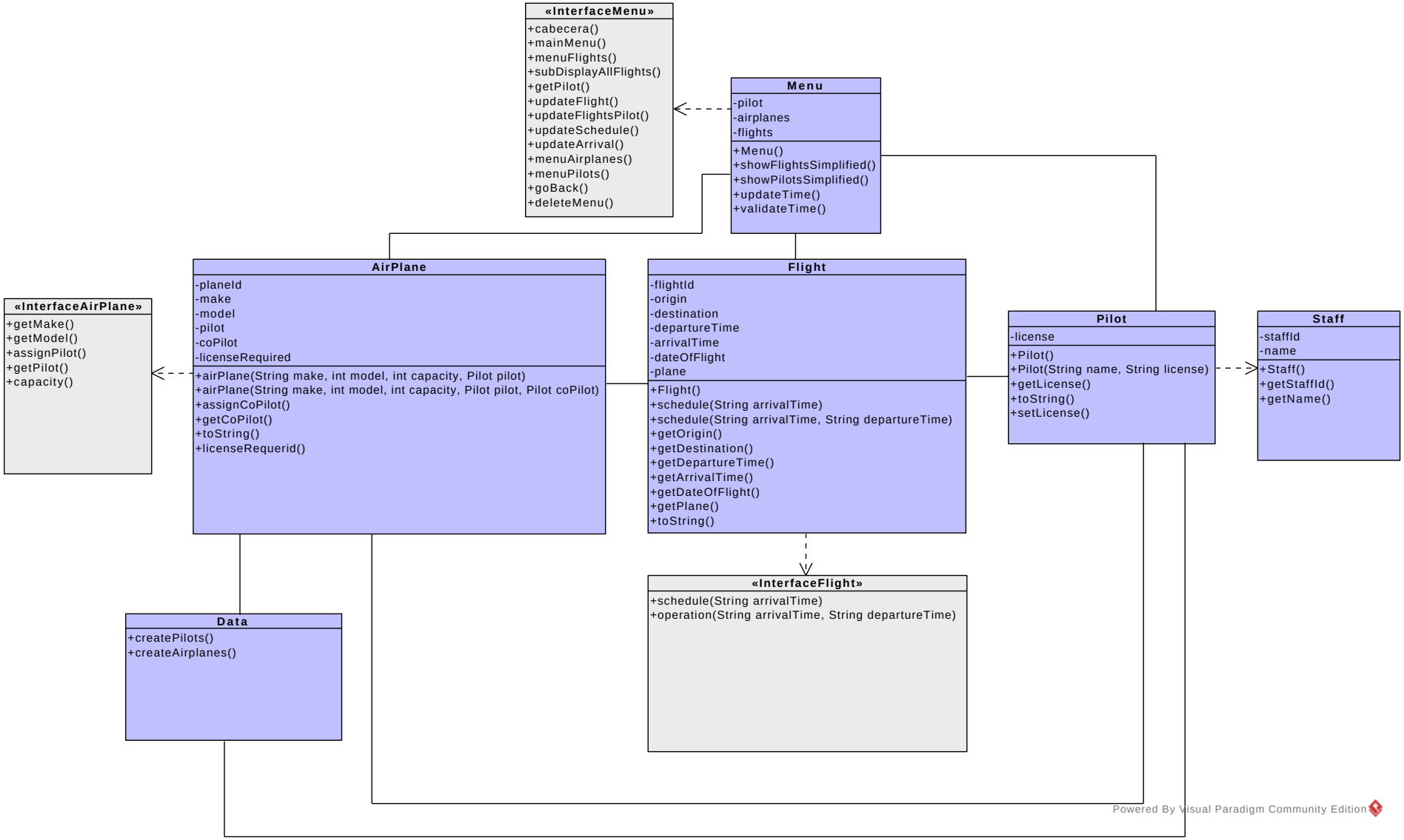

Figure 3.1: Class diagram

4 Work plan and allocation of roles and responsibilities to each member of the development team

One of the key points of this project has been the planning of a team work. Therefore, after reasoning and decided the Class design, we build an Interface for each Class. The interfaces contain a collection of methods for each class. This allows each member of the team to start coding a class but following the pattern specified by the interface. In other words, we used interfaces to achieve loose coupling and thus facilitate teamwork.

The responsibilities were allocated in the following way:

- **Asmer:**
  - Design of the interfaces.
  - *Staff*, *Pilot*, *AirPlain* and *CCTAir*.
- **Miguelantonio:** *Flight*, *CCTAir*, *Data*.
- **Adelo:** *Menu* and *CCTAir*.

It is important to point out that a meticulous teamwork was necessary to reach the requirements of the project. A set of group work sessions were crucial in order to discuss issues and decide key points in the development process.

Declaration

We hereby declare that all of the work shown here is our own work.

Student Number: Miguelantonio Guerra

Student Number: Asmer Bracho

Student's Name: Adelo Vieira

Date: April 1, 2018

Bibliography

STACKOVERFLOW.COM : *Best practice to create two-level deep selection menu in Java console*. URL https://stackoverflow.com/questions/30191614/best-practice-to-create-two-level-deep-selection-menu-in-java-console.