Airline

Create an application to manage an airline organization. The application must allow creating aircrafts of different type, flights and booking tickets. All the classes must be in the package "**`Airline`**".

R1 - Fleet

The system works though the facade class **`Airline`**. The *`initializer`* accepts the name of the airline.
To define a new aircraft, we can use the function **`newAircraft()`** of class **`Airline`** that accepts as arguments the model of the aircraft, the *`distance range`* defined as an *`enum`* variable, the number of rows in Business Class, the number of rows in the Main Cabin and the number of seats per row. We can assume that the number or seat per rows of the Business and Main class are the same. The model of the aircraft can be considered unique. If the aircraft's model already exists, error **`InvalidAircraftModel`** is thrown.
Thanks to the *`distance range`*, three types of planes can be defined:

- Plane for regional flights: *`Short`*
- Plane for medium-distance flights: *`Medium`*
- Plane for long-distance flights: *`Long`*

To retrieve some information about the fleet we can use the function **`getAircrafts()`** which returns a list of aircrafts, sorted by *`distance range`* (Long - Medium - Short) and in alphabetic order.
If a distance range is passed as a parameter, the function **`getAircrafts()`** returns the list of the aircrafts of that class sorted by model. If the number or seats is passed as a parameter, the function **`getAircrafts()`** return the list of the aircrafts with a number of seats greater or equal to the value passed as parameter sorted by number of seats and by model.
To get info about a specific aircraft, we can use the function **`getAircraft()`** that accepts the model of the aircraft and returns the corresponding aircraft. If the model is not present then error **`AircraftNotFound`** is thrown.

R2 - Flights

To define a new flight, we can use the function **`newFlight()`** of class **`Airline`** that accepts as argument the code of the flight, the aircraft, the airport and the date and time (UTC[1]) of departure[2], the airport and the date and time (UTC) of the arrival and the distance expressed thanks an *`enum`* variable. We can assume that the code of the flight is

[1] Coordinated Universal Time

[2] **Hint:** Use NSDate to set date and time. See documentation for more information.

unique. If the aircraft is not appropriate to the distance then the error **`DistanceNotSupported`** is thrown.
To retrieve some informations about the flights we can use the function **`getFlights()`** which returns a list of flights, sorted by date, time and finally by name of the departure and arrival.
To get info about a specific flight, we can use the function **`getFlight()`** that accepts the code of the flight. To delete a flight we can use the function **`deleteFlight()`** that accepts the code of the flight. In the last two functions, if the code is not present then the error **`FlightNotFound`** is thrown.

R3 - Booking

To book a flight, we can use the function **`bookFlight()`** of class **`Airline`** that accepts the code of the flight, the number of people and the *`classSection`* (*`Business`* or *`Main`*).
If there are no seats available then the error **`NoSeatAvailable`** is thrown.
If there is no Flight with that specific flight code, error **`FlightNotFound`** is thrown.

R4 - Statistics

The function **`fullFlights()`** returns the flights in witch every seat was booked, sorted by date, time and finally by flight code.
The function **`averageBooking()`** returns the average of seats booked for the flights in the schedule. There are three other versions of this function:

- passing as a parameter a *`classSection`* (*`Business`* or *`Main`*), the function returns the average of booked seats for that class
- passing as a parameter a distance range (*`Short`*, *`Medium`* or *`Long`*), the function returns the average of booked seats for that distance range
- passing as a parameter an aircraft model, the function returns the average of booked seats for that specific aircraft model. If aircraft model doesn't exist, error **`InvalidAircraftModel`** is thrown.

The function **`mostUsedAircraft()`** returns a tuple containing the most used aircraft in the fleet and the number of time it was used.
The function **`getStatus()`** accepts the flight code and the current time and returns an *`enum`* variable called *`Status`* that represent the status of the flight respect the current time. If the current time is after the departure time *`WillTakeOff(Int)`* is returned (*`Int`* represents the difference of time of the current time and the departure time expressed in minutes). If the current time is the same of the departure time *`TakingOff`* is returned.
If the current time is before the departure time *`DidTakeOff(Int)`* is returned.