Airline Manager

Turma 9 Grupo 69

- André Barbosa up202007398
- Guilherme Almeida up20200886
- José Luís Rodrigues up202008462

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

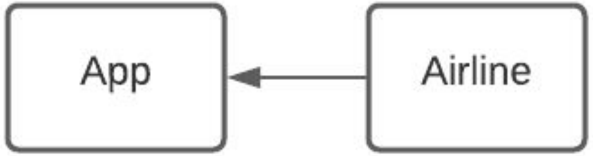
Context

A new airline has joined the market and needs a management system to coordinate their operations.

The system should keep information about airports, planes, flights and passengers. All the planes need to be repaired and cleaned so the program should also be able to deal with all the maintenance planning. Passengers can acquire tickets and check in to the flights. When a passenger is carrying luggage, it needs to be sorted into the carts that can take it to the plane. It should also store information about the airports, like the transports around.

Program

Our solution to this problem was a CLI (Command Line Interface) that allows you to manage all the airline information. Each component (Airports, Flight, Plane, ...) were implemented separately in different classes and were brought together in the Airline Class, which is the main controller of the program. Then, to make it into a user-friendly interface we created the App, which is independent to the Airline.

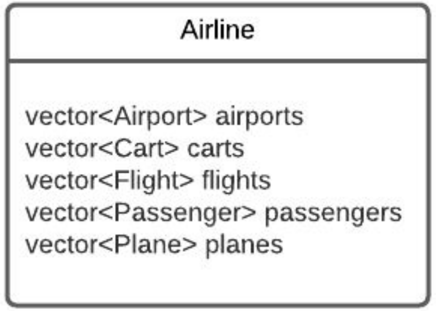


Algorithms

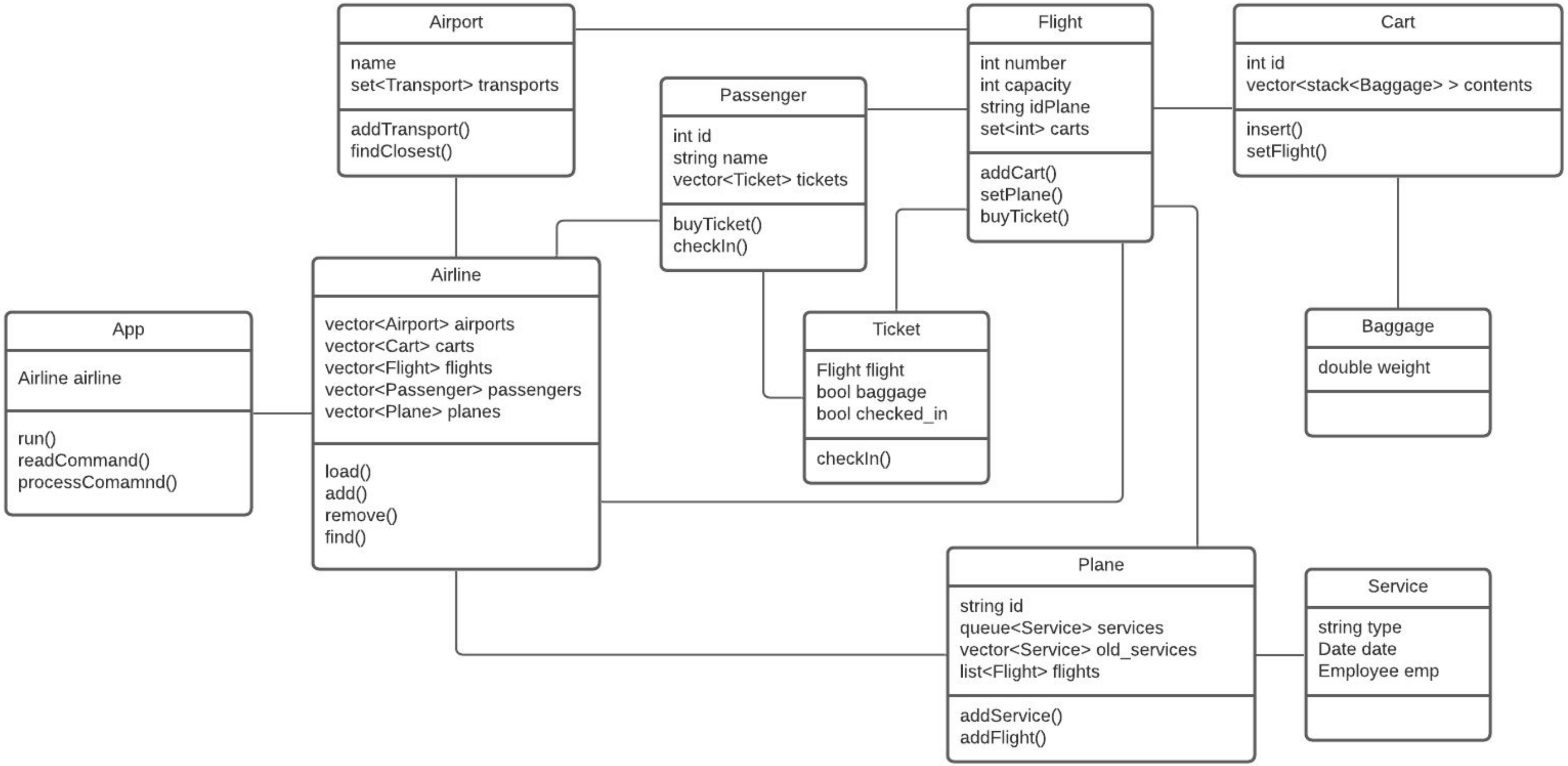
In the Airline class, a vector is kept for airports, carts, flights, passengers and planes. When the data is loaded or when the user changes the order, the vectors are sorted using the STL **sort()** method. In other operations, like when editing the vectors, we can assume they are already sorted, so we used **insertion sort** to update them.

Because the vectors are sorted, we can use **binary search** to find the elements we are looking for.

We used the STL **set** (binary search tree) to save the transports in order of distance to the airport.



Class Diagram

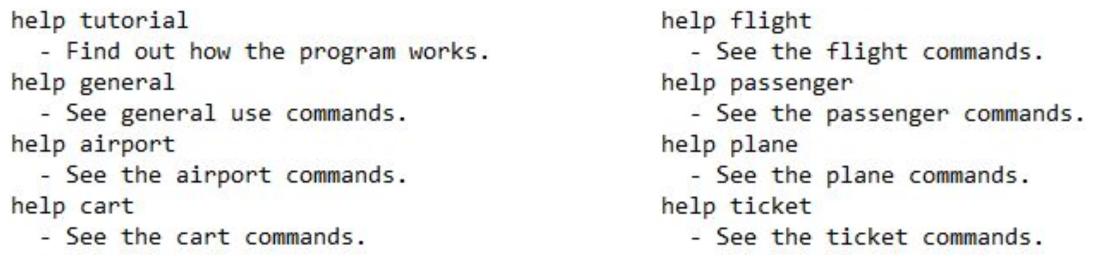


Usage

The program works through the console. When you start Airline Manager the following message is prompted:

Welcome to Airline Manager. Use help to get started.

When help is used, the program outputs a series of commands so you know what to do.



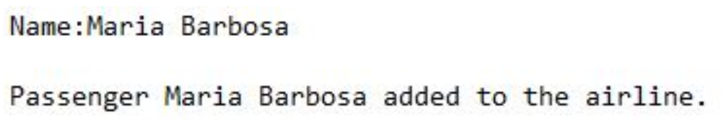
```
help tutorial
  - Find out how the program works.
help general
  - See general use commands.
help airport
  - See the airport commands.
help cart
  - See the cart commands.
help flight
  - See the flight commands.
help passenger
  - See the passenger commands.
help plane
  - See the plane commands.
help ticket
  - See the ticket commands.
```

Example

Let’s try to buy a ticket to a new passenger using Airline Manager.

First, we should add the passenger:

```
>passenger add 37894532
```

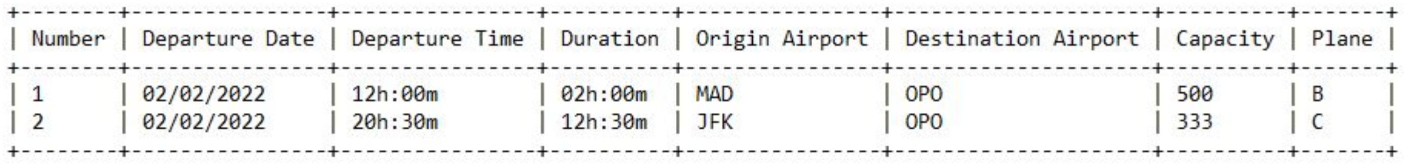


```
Name:Maria Barbosa

Passenger Maria Barbosa added to the airline.
```

The passenger is now on the airline. Now, let’s choose a flight to buy a ticket from:

```
>flight display
```

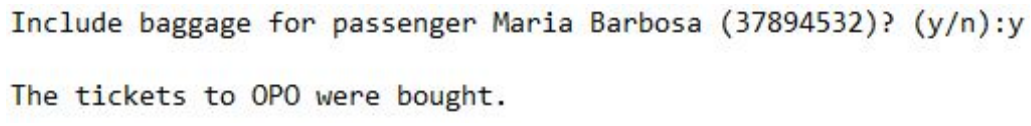


```
+--------+----------------+----------------+----------+-----------------+---------------------+----------+-------+
| Number | Departure Date | Departure Time | Duration | Origin Airport  | Destination Airport | Capacity | Plane |
+--------+----------------+----------------+----------+-----------------+---------------------+----------+-------+
| 1      | 02/02/2022     | 12h:00m        | 02h:00m  | MAD             | OPO                 | 500      | B     |
| 2      | 02/02/2022     | 20h:30m        | 12h:30m  | JFK             | OPO                 | 333      | C     |
+--------+----------------+----------------+----------+-----------------+---------------------+----------+-------+
```

Example

From all the flights, we chose to buy a ticket for flight number 1 flying to Oporto. So now we buy the ticket:

```
>ticket buy 1 37894532
```



```
Include baggage for passenger Maria Barbosa (37894532)? (y/n):y

The tickets to OPO were bought.
```

That’s it! Now you can also choose to check in that passenger using `passenger checkin`.

Functionalities

Each data group has fully operational **CRUD** operations (add, display, edit and remove). You can also use find to locate the data you are looking for.

It is also possible to change the displaying order for flight, passenger and plane.

Passengers can be searched by their names. In this case, the program finds all matches for the searched name.

Flights have a partial display functionality where you can search for flights in a date frame.

Command Interface

The implemented commands are well structured, easy to understand and very responsive. We made sure you wouldn’t get stuck trying to use them. The program is easy to use because it provides continuous explanation of what’s going on.

The command structure makes the program fast to use but at the same time very visual.

We implemented a quick tutorial that helps you get started using the commands. Additionally every time you don’t know what to do, help will guide you.

Files

To store the information we used csv files. This way, each row holds different information needed for the program, as such:

- *airport.csv* - Name, (Transport Type, Transport Distance, Transport Schedule)*
- *carts.csv* - IdCart, Carriages, Piles, Bags, Flight, (Baggage)*
- *flights.csv* - Number, Departure Date, Departure Time, Duration, Origin Airport, Destination Airport, Capacity, Plane
- *passengers.csv* - Id, Name, (Flight, Baggage, Check In)*
- *planes.csv* - Id, Model, Capacity

An example from *passengers.csv*:

```
30583893,Maria Barbosa,12,1,0
```

Difficulties

During the development of the program, we encountered some difficulties organizing the code and making design decisions. Sometimes, we found it hard to find where algorithms could be used and which data structure was best.

Some things that were hard at first, like dividing tasks and working together turned out to be a learning opportunity.