University “Politehnica” of Bucharest

Faculty of Electronics, Telecommunications, and Information Technology

**Flight reservation system**

**Object Oriented Programming**

Student: Duta Alexandru Vlad

Group: 413G

2023

**Introduction**

The current aplication simulates a basic flight reservation system for an airline.It is designed to store information regarding the flight and its passengers. Each passenger can enter their details for reservation such as name, age,weight(kg),destionation and at the end they can select for business class or economy class, and for price of ticket too. All those actions are associated to a specific passenger id ,which allows the airline to receive new passenger, to remove seat or change it between passengers, To interact with the user, it was built a command line interface structured as a numbered menu. The operator can select an option by entering its corresponding number and pressing enter. In the command line window, there will be shown both the application data, as well as information with respect to objects creation and deletion to emphasize the back-end architecture.

To build the program there were used two main classes: Passenger, and Airplane and two child classes :BusinesssPassenger and EconomyPassenger. The application was programmed such that Passenger is the base class, from which BusinessPassenger and EconomyPassenger inherits its elements. The class Passenger is mainly used to define the passengers and create a representation of them(id, age ,name, weight). The Airplane class manages the passengers on an airplane. It keeps track of the total number of seats and the number of seats that are currently occupied. It maintains a list of passengers using a vector.

When running the aplication, the first step will be to create a passenger , meaning adding the general data(id,age, name, weight, destionation) as well as their choose from business or economy class.Once the data is in place, the user can generate random passengers or create manually a new one. The user can perform different action, from displaying passenger detailts, displaying all passenger, to swaping seat between them, or to remove a passenger. To ease the understanding a set of instructions was incorporated into the main menu of the application.

**Program Structure**

The build aplication is structured as follows:

1. A base called Passenger which contains generic information about passengers.

This class has the following attributes:

* **id**: An integer representing the passenger's ID.
* **name**: A string representing the passenger's name.
* **dest**: A string representing the passenger's destination.
* **age**: An integer representing the passenger's age.
* **weight**: A double representing the passenger's weight.
* **ticketPrice**: A double representing the price of the passenger's ticket.

Member Functions:

* **getId()**: A getter function that returns the passenger's ID.
* **displayDetails()**: A function that displays the details of the passenger, including the ID, name, age, weight, destination, and ticket price.

2. A class called BusinessPassenger which inherits the elements of Passenger.

In this constructor, the derived class **BusinessPassenger** is initialized by calling the base class **Passenger** constructor with the provided arguments. The **Passenger** constructor initializes the **id**, **name**, **age**, **weight**, **dest**, and **ticketPrice** data members of the **Passenger** class with the provided values.

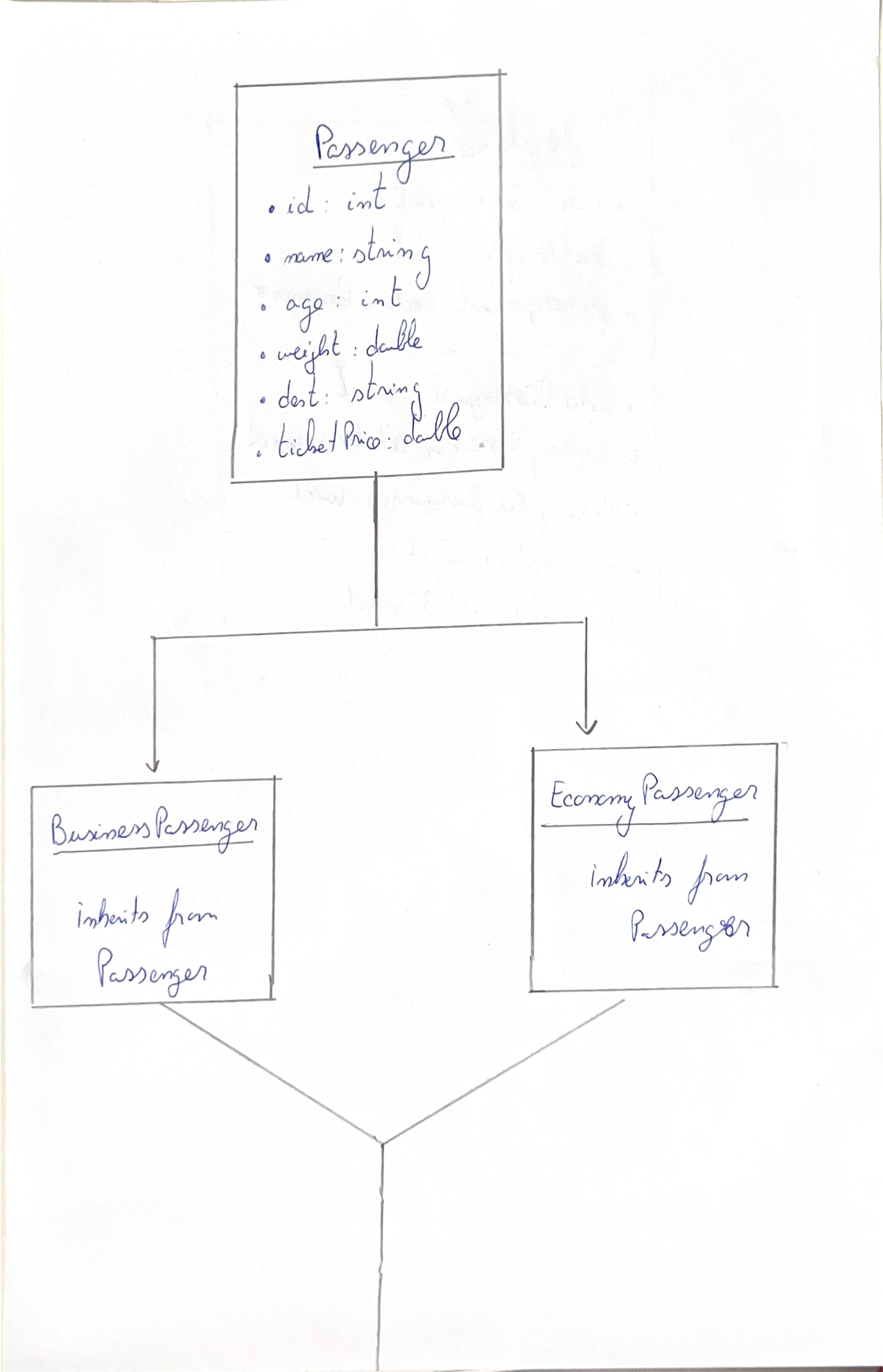
3. A class called EconomyPassenger which inherits the elements of Passenger.

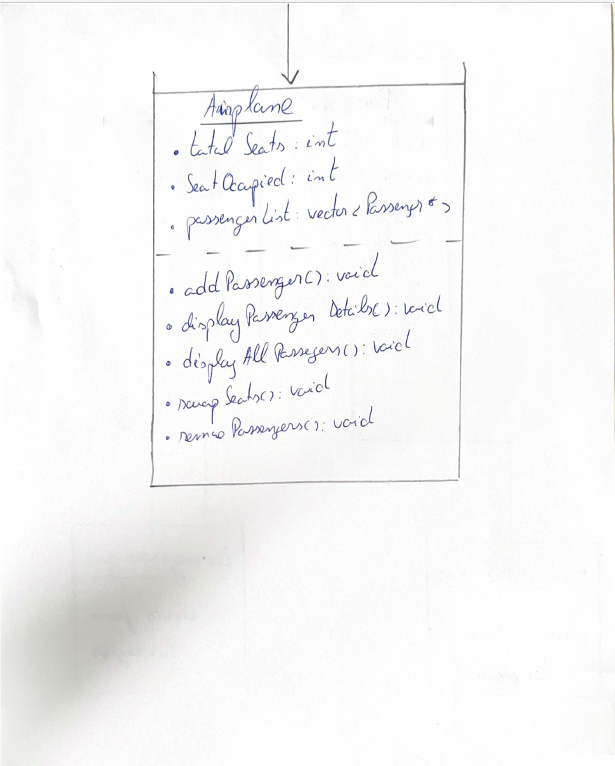
In this constructor, the derived class **EconomyPassenger** is initialized by calling the base class **Passenger** constructor with the provided arguments. The **Passenger** constructor initializes the **id**, **name**, **age**, **weight**, **dest**, and **ticketPrice** data members of the **Passenger** class with the provided values.

4. The **Airplane** class represents an airplane in the airline reservation system. It is responsible for managing the passengers on the airplane and performing operations related to passengers:

* Manages the total number of seats on the airplane (**totalSeats**) and the number of seats currently occupied (**seatsOccupied**).
* Keeps track of the list of passengers on the airplane using a vector of **Passenger\*** objects (**passengerList**).
* Provides methods to add a passenger, display passenger details, display all passengers, swap seats between passengers, and remove a passenger.
* The **addPassenger()** method adds a **Passenger\*** object to the **passengerList** vector if there are available seats on the airplane. It increments the **seatsOccupied** counter and displays a message indicating whether the passenger was successfully added or if the airplane is already full.
* The **displayPassengerDetails()** method takes a passenger ID as input and searches for the corresponding passenger in the **passengerList**. If found, it calls the **displayDetails()** method of the passenger to display their details. If not found, it displays a message indicating that the passenger was not found.
* The **displayAllPassengers()** method iterates over the **passengerList** vector and calls the **displayDetails()** method for each passenger, effectively displaying the details of all passengers on the airplane.
* The **swapSeats()** method takes two passenger IDs as input and searches for the corresponding passengers in the **passengerList**. If both passengers are found, it performs the logic to swap their seats. The actual implementation of the seat swapping logic is missing in the provided code.
* The **removePassenger()** method takes a passenger ID as input and searches for the corresponding passenger in the **passengerList**. If found, it removes the passenger from the **passengerList**, deletes the passenger object to free the allocated memory, decrements the **seatsOccupied** counter, and displays a message indicating that the passenger was removed. If the passenger is not found, it displays a message indicating that the passenger was not found.
* The destructor **~Airplane()** is responsible for cleaning up the dynamically allocated **Passenger** objects in the **passengerList** vector by iterating over the vector and deleting each passenger object. It also clears the **passengerList** vector.

**Graphical Representation**



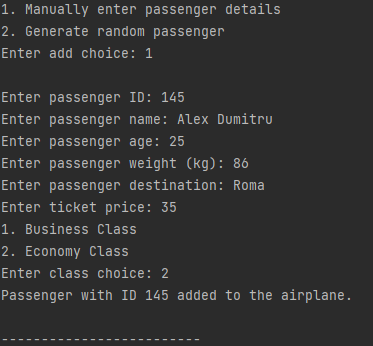


**Application Demonstration**

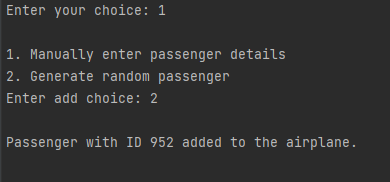
In the image below we can observe the application menu which automatically display when you run the program and waits for a selection to be made. We start by adding a passenger, so we press 1 and enter key.



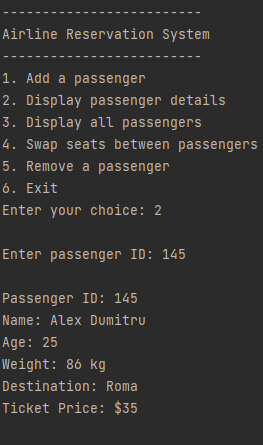
When creating a passenger profile ,the program ask for general details about the passenger (id,age,name,weight) and the flight(destination and ticket price).At the end of passenger configuration, the program ask for a decision between business or economy class, which can be decided by pressing 1 or 2 and enter key.



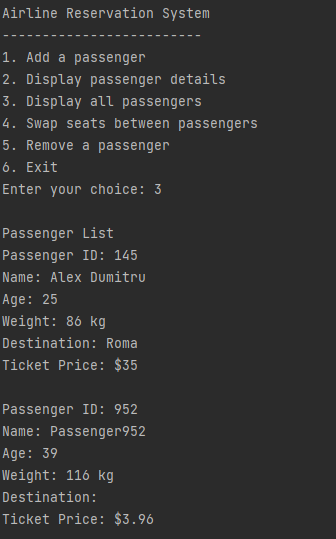
Also the program can generate a random passenger by pressing 2 and enter key , and it will be sort by the id.



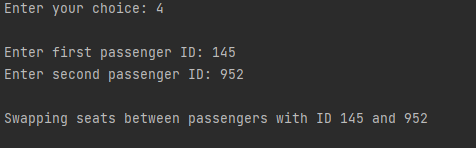
The program returns to the main menu every time a task is done. In the menu you can see passenger details by pressing 2 and the enter key. Then the passenger id is required to show the details. The program will show the details and return to the main menu.



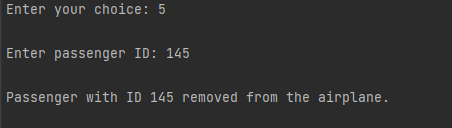
If you want to see all the passengers press 3 and enter key, and all the passenger details will pop up(for us it will be showed our manually entered passenger and the random generated one)



Back in the main menu you can also swap seat between passenger by pressing 4 and enter key. Both passengers id are required for this change.



Last thing you can do in the main menu is removeing a passenger. Here the id is also necessary for this operation.



**Conclusion**

In conclusion, the provided code represents a simplified airline reservation system. It consists of three classes: **Passenger**, **BusinessPassenger**, and **EconomyPassenger**, representing different types of passengers, and an **Airplane** class representing an airplane with a certain number of seats.

The **Passenger** class holds common attributes such as ID, name, age, weight, destination, and ticket price. It provides methods to retrieve and display passenger details.

The **BusinessPassenger** and **EconomyPassenger** classes are derived from the **Passenger** class, inheriting its attributes and methods. These classes serve as specialized passenger types, distinguishing between business class and economy class passengers. They don't have any additional attributes or methods beyond those inherited from the **Passenger** class.

The **Airplane** class represents an airplane with a specified number of seats. It keeps track of the total number of seats and the number of seats occupied. It maintains a list of passengers on board, implemented as a vector of **Passenger** pointers. The class provides functionality to add passengers, display passenger details, display all passengers, swap seats between passengers, and remove passengers.

Overall, the code allows users to interact with the airline reservation system by adding passengers manually or generating random passengers, displaying passenger details, viewing all passengers, swapping seats between passengers, and removing passengers. It provides a basic structure for managing passengers and their information within the context of an airplane.

**Annex – Code Files**

