UNIVERSITÉ DU LUXEMBOURG

Programming 1

Ali Osman TOPAL (aliosman.topal@uni.lu)

Lab 6 – Object-Oriented Programming

2021–10–25

Submission (Preliminary: 07/11, Final: 14/11)
The mandatory exercises for **Code Submission** are **1, 2** (marked with an ❗). All other exercises on this sheet are optional but still highly recommended! The **Explainer Video** 🎥 for this sheet must be realized on **Exercise 1 ("Museum")**. A Flipgrid invitation link will be posted on Moodle.

Exercise 1 – Museum ❗🎥 #Inheritance #Encapsulation # Polymorphism #Extends #Public #Private

The Luxembourg History Museum needs a ticket system to manage entrances and costs. The particular aspect of the system is that the ticket depend on the museum pavilions that the visitor wish to visit.

1° The museum pavilion is represented by the class **`Pavilion`**. This class has the following attributes: `String name` and `double price`. Both attributes are private and compulsory at the moment you create an instance. Implement the class with the appropriate constructor and get methods.

2° Implement a class **`Ticket`** with the following attributes: `UUID ticketID`, `Date emissionDate`, and an array of `Pavilion` class. ticketID must be unique and automatically generated, and also emissionDate must be automatically initialized with the current date when a ticket instance is created. An array of `Pavilion` class is compulsory at the moment you create an instance. The cost of a ticket is calculated by summing the price for each pavilion and by applying the following discount. For the single tickets, the full price is paid for the first three pavilions. For each further pavilion a 30% discount is applied.

3° Implement a class **`GroupTicket`** inheriting from the `Ticket` class. The `GroupTicket` class has an additional attribute `int n` which represents the number of visitors in a group and which is private and required in addition to the array of Pavilion class when you create an instance. The cost of the ticket is; if the group is between 6 and 12 (included) a total discount of 30% is applied to the overall price whatever the number of pavilions. If the group is bigger than 12, the discount is 40%.

Both `Ticket` and `GroupTicket` classes have String toString() method prints the ticket details on the console as shown in the sample output (`toString()` method should be implemented without redundancy in the `GroupTicket` class).

4° Write a test program that creates some instances of the implemented classes.

Sample Output

```
— Welcome to Luxemobourg History Museum —
        Ticket ID: 6b6b0430-1a92-48ce-b09a-58681a3b8645
        Date: Wed Oct 27 15:48:04 CEST 2021
        You can visit: pav2 pav3 pav4 pav1 pav5
        Total cost is: $356.0
```

```
        –** Group Ticket **–
— Welcome to Luxembourg History Museum —
        Ticket ID: ab9c826c-a542-42f3-baa8-6472fcaa5639
        Date: Wed Oct 27 15:48:04 CEST 2021
        You can visit: pav4 pav8 pav7 pav6 pav9
        Total cost is: $266.0
```

Exercise 2 – ! Air Travel

1° Implement the classes of the UML class diagram shown in Fig. 1. If you consider further fields or classes useful, extend the class hierarchy accordingly.

> ✎ **Code Generation**
>
> VS Code provides support for generating code (e.g. constructors, getters & setters, ...) based on declared fields (attributes). This can considerably improve your development efficiency, taking away the burden of manually writing basic functionality. Of course, this does not mean you could (or should) not alter the generated code with custom business logic (e.g. sanity checks in setters, ...).
>
> To generate code in VS Code, right-click in the code editor and select Source Action... . From there, choose, e.g., Generate Getters and Setters... and select the attributes for which they should be generated. Try also Generate Constructors... .

2° Which class(es) should manage the participation of a passenger or a pilot in a (set of) flight(s)? Think about an appropriate way to implement this functionality and then realize it. What are the advantages and disadvantages of your proposed solution[1]?

3° Write a main program that creates some instances of the implemented classes.

Exercise 3 – Facebook

In this task, you will create a Facebook-like implementation of profiles. Profiles have an ID, a name and a set of posts. Posts can be created and the posts on the profile's timeline can be shown to an observing user.

[1] As already mentioned in the lecture, there is often no single correct design.

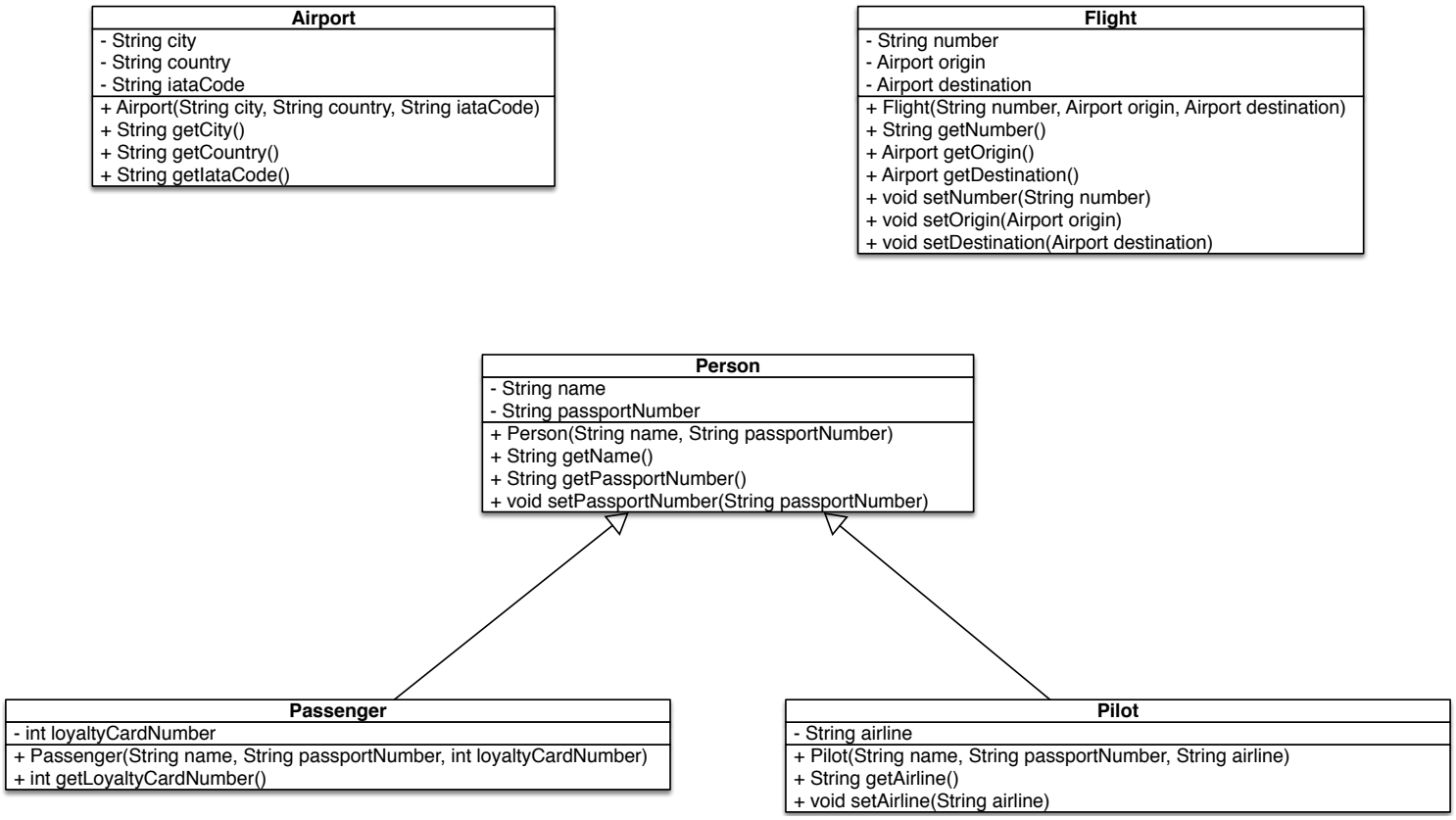

Figure 1 – Air Travel UML Class Diagram

There are two types of profiles: regular users and pages. Users hold a set of friends. New friends can be added. The timeline is only shown to users you are friends with. Pages hold a like counter which gets incremented every time a user likes a page. The timeline of a page can be shown to everyone.

Design and implement a set of classes according to these requirements. Make sure that a profile has to be either a regular user or a page. Take care of the changing behavior with respect to the "privacy" of a timeline for the different profile types. Finally, write a main program to test your implementation.