Programming 1

Christian Grévisse (`christian.grevisse@uni.lu`)
Gilles Neyens (`gilles.neyens@uni.lu`)

Lab 6 – Recursion & Object-Oriented Programming

2017–11–14

Part I – Recursion

Exercise 1 – Fibonacci reloaded

In Lab 3 – Exercise 7, you implemented the Fibonacci sequence based on loops. Now, implement it based on **recursion**. As a reminder, note that the Fibonacci sequence is defined as:

$$\begin{cases} \mathcal{F}_1 & = 1 \\ \mathcal{F}_2 & = 1 \\ \mathcal{F}_n & = \mathcal{F}_{n-1} + \mathcal{F}_{n-2} \quad n > 2 \end{cases}$$

Exercise 2* – Memoization

Looking at the *call graph* of the recursive calculation of $\mathcal{F}_6$ as in figure 1, we can see that there is quite some redundancy, as many calculations are done several times, which is quite inefficient. In addition, the higher n, the more the stack will be filled for the calculation of $\mathcal{F}_n$. As mentioned in the lecture, this may lead to a `java.lang.StackOverflowError`.

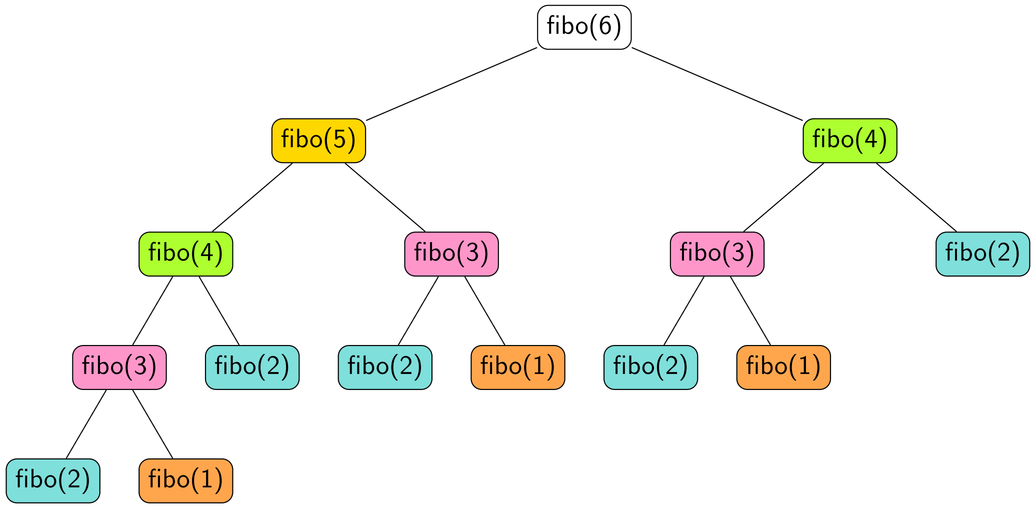

Figure 1 – Call graph of $\mathcal{F}_6$

A common optimization technique is *memoization*, where already calculated values are cached such that the result can be retrieved and reused later on, without needing to recalculate them (and thus waste CPU cycles).

– Extend your implementation from exercise 1 by enabling memoization.

– Compare the execution times between using and not using memoization. You might need `System.nanoTime()`.

Part II – Object-Oriented Programming (Encapsulation, Inheritance, Polymorphism)

Exercise 3 – Air Travel

#Inheritance #Encapsulation #Extends #Public #Private

1° Implement the classes of the UML class diagram shown in figure 2. If you consider further fields or classes useful, extend the class hierarchy accordingly.

> ✎ **Eclipse Code Generation**
> Eclipse offers you quite some menu items to generate 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, ...).
>
> Have a look at what Eclipse offers you in the Source menu, foremost Source › Generate Getters and Setters... as well as Source › Generate Constructor using Fields....
>
> When creating a new class, the dialog also allows you to specify the superclass (by default `java.lang.Object`) and create a constructor based on the superclass' constructor.

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 4 – 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.

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.

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

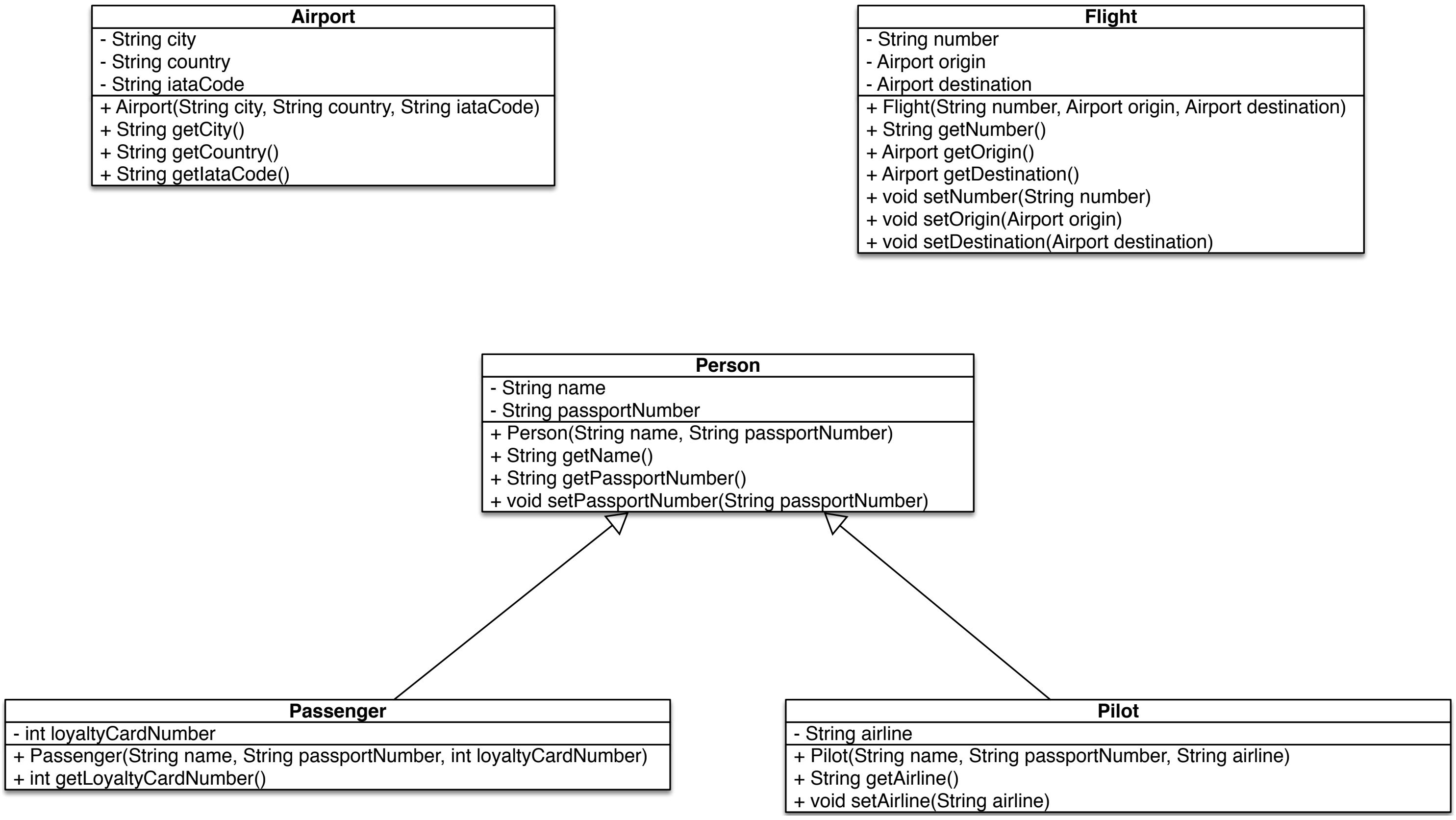
Airport
- String city
- String country
- String iataCode
+ Airport(String city, String country, String iataCode)
+ String getCity()
+ String getCountry()
+ String getIataCode()
Flight
- String number
- Airport origin
- Airport destination
+ Flight(String number, Airport origin, Airport destination)
+ String getNumber()
+ Airport getOrigin()
+ Airport getDestination()
+ void setNumber(String number)
+ void setOrigin(Airport origin)
+ void setDestination(Airport destination)
Person
- String name
- String passportNumber
+ Person(String name, String passportNumber)
+ String getName()
+ String getPassportNumber()
+ void setPassportNumber(String passportNumber)
Passenger
- int loyaltyCardNumber
+ Passenger(String name, String passportNumber, int loyaltyCardNumber)
+ int getLoyaltyCardNumber()
Pilot
- String airline
+ Pilot(String name, String passportNumber, String airline)
+ String getAirline()
+ void setAirline(String airline)

Figure 2 – Air Travel UML Class Diagram