JAVA2 exercise. A Bad Dream.

Pieter van den Hombergh

6th March 2017

It is only a dream. . . . ..

Olifantys is a University in a galaxy not far from here. They program in Java and use Open Source. Must have to do with their Motto, Think Smaller. Anyway, you wake up in said galaxy, student at Olifantys, in an institute named Sebi. No idea what that name might mean. They say you claim to be smart. So you better make it count. . .

Problem of Galactic Proportions

Olifantys wants a design and implementation of a new information system to replace their current administrative system, called RetroGression. They also made the decision to make a clean cut with the past, thereby preventing design "*features*" to creep into the new application. By the way, there is a competition for the name of the new product, and its is still open, so you might want to participate. Of course names which already are TradeMarks® are a no go.

. . . But real enough to be software engineering exercise

To make a long story short, here is the job you find at your desk, as soon a you open your eyes: Start with a sketchy class design in UML, then implement it in Java. The important part is that you make your design such that it is little work to implement, easy to explain and also avoid redundancy. This will help to get your design past the Olifantys bureaucracy, known for their thick skulls. . . . You have been warned.

Some tips you might remember from your Java classes in our Galaxy, the Milky Way: You can use Java 8 features. Use interfaces and or abstract classes to reuse parts that would or could be similar. It will impress the bureaucracy.

The design should accommodate persons, the supertype of all other kinds of beings, who are born at some date, and have (a) sex (or hope to have it), but gender neutral is also supported. Persons optionally can have a partner. Olifantys is liberal here and allows partners of the same or other sex. Not having sex is never an issue anywhere, so neither it is at Olifantys.

Possible roles in this play following kind:

Student : Happens to be the kind *you* are. You can think of your own attributes (instance variables, fields, method, you name it). Some of these attributes are shared with Persons of other varieties, such as

Teacher : You meet all too often,

Manager : You seldom see, but they are said to be important

Administrator : who keep annoying students and teachers alike with all kinds of details which are seldom of immediate importance

Director : Someone who needs to be taken seriously, as an institute. Fends for the institute's interest with other directors.

Combos : such as Adminstrator-Manager, Teacher-Manager and Student-Assistant.

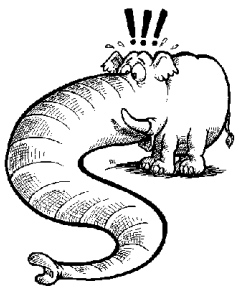

Figure 1: Olifantys Logo, not nearly as fishy as some Logos.

Roles can be combined: For instance in the administration we have a manager, the administration-manager, which has all the functionally of both administrator and manager.

Something to study

The students study, at least, that is the idea. Your task is to model the study topics as `enum`s. Model the modules like JAVA2, SEN1, MOD1, BUA etc as enum constants. Make sure it fas such attributes as name, description (one line of text), credits, number of weekly hours etc. Make the students can study the topics, the teachers can teach and the administrators record the progress of the students. How would

you map the grades a student receives? Can a module have sub-modules, and in what order must you define the them in the enum type?

Lets party

Students want to throw a party at the DomboBar, for no particular reason. This epic contains the following user stories:

- As a *student*, when I want to throw a party, I want to **make a group** in which can use to put all invitees in.
- As a *bar owner* **I want to put** all the drunken employees in a **taxi-cab**, to get them safely home.
- As a *party tiger*, I want to **sing seesAJollyGood** to **everyone** that has birthday this month.

Oberlings, Underlings

Managers are a special species. They have all day to manage their underlings[1]. `Underling` and at the same time `Oberling`[2] might be new words: Underlings are those that are managed by the Oberlings and must say yes Sir, yes Ma'am to their Oberling and my not decide on their own. Silly culture, no? By the way: The director is the manager of managers, you can model that too, right?

Typically managers have benefits: They get payed more, but do not have to spend that much, because their car is company owned, and their parking lot is reserved for them, no fee. Electric car, so no fuel bill either. There is one exception though: The `TeacherManagers` do not have such benefits. They and their predecessors never have had the time to fight for such extravaganzas. Nor are true Teachers willing to strike because it makes their heart bleed for those poor Students that would be left uneducated.

However, Teachers at Olifantys have the benefit of having underlings: Their Students, thereby being Oberling of sorts. ManagingTeachers doubly so because they have Students as well as underling teachers. Make sure you model that too, because your design and implementation will be scrutinised by *would be* TeacherManager Ebeneezer. Some teachers have not noticed it yet, but they *must share* their underlings, so it is not always clear what students will do and who they will listen to. This is commonly believed to be beneficial for the student in the end. It makes the student more resistant to utter confusion because of earlier exposure.

Of course the student is the sod of sods, because he/she not only underlies a teacher but all teachers. Time to get on top, you poor sod.

There is light: Even students can have underlings. This is not officially condoned, but can be modeled. Such Oberling[3] students are either called Bully, but also SmartAss, ClassRepresentative and StudentAssistant is common.

We think the path to revenge is also quite clear from the model: Achieve director-hood while that Ebeneezer still teaches or in a wise wizards words: fill the directors shoes[4].

Tech talk

Keep the initial design simple, because any improvement in a class higher up in the class hierarchy will automatically also be available in the lower classes.

Given the class diagram, explain it in your own words in an asciidoc file in your repository under week2. In particular explain the relations between person and student and person and employee.

You can start with the olifantys project in a zip file on the website. From the given class diagram on page 2, design the classes and their relationships by choosing methods and method names and document those. Write what they do, not how. Put the methods in classes or interfaces.

From the that design, per method, write the tests to the method and implement the method. Preferably work test driven.

Implement the user stories with generics using upper (extends) and lower (super) bounds where appropriate.

Hint: implementing a method in a class A dictated by an interface I, whose method work is already implemented in some class C (or known to be implemented) can be very easily be made to work, by delegating the work to and instance of C, by making such an instance field of class A.

It is best if you do this in helper classes which are not visible outside the Olifantys package. For instance if you want to have all the code you need in implementing a manager interface, you might want to use a helper class Oberling, which implements exactly those methods that support what you can do with underlings: List them, sack them, add them, count them, give them grades (or salary/raise if employee). Other tip: Start with a simple design, then gradually add *generics* to the soup and keep stirring/testing.

[1] Normal English would say subordinates. We are on another planet, remember...
[2] Superior would be the English word.
[3] You claim to brightness should help you to understand this word too.
[4] Idea shamelessly stolen from Sir Terry Pratchett[R.I.P. 2015] of Discworld fame

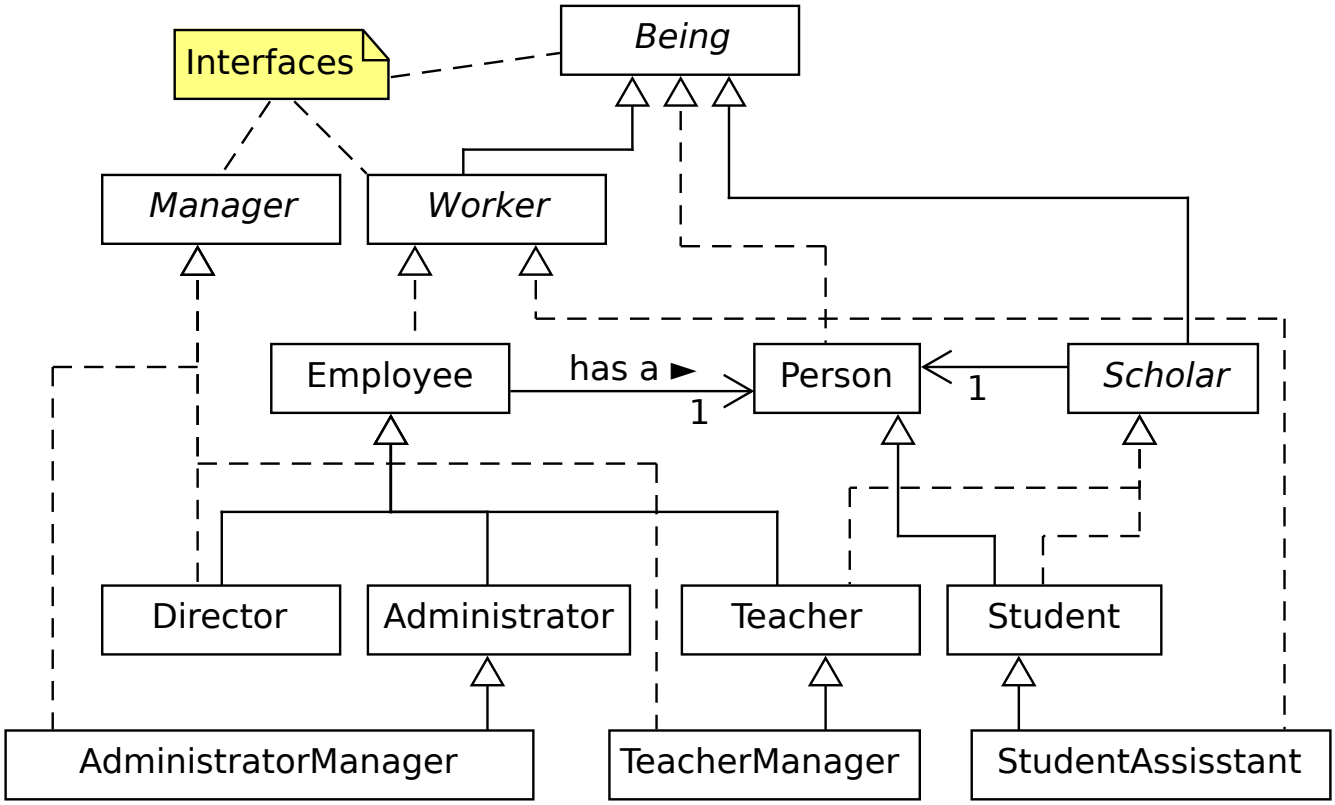

Figure 2: Class diagram

Must stop stop typing now. Coffee is brewn and I must mail my Oberling.

```
End of episode 1.  Stay tuned.
```

Disclaimer

All characters and events in this story are entirely fictional.

Any likeness or similarities between actual institutes, organizations and people are purely coincidental.