Software-Engineering 2

Prof. Dr.-Ing. Gerhard Wanner
Email: wanner@hft-stuttgart.de

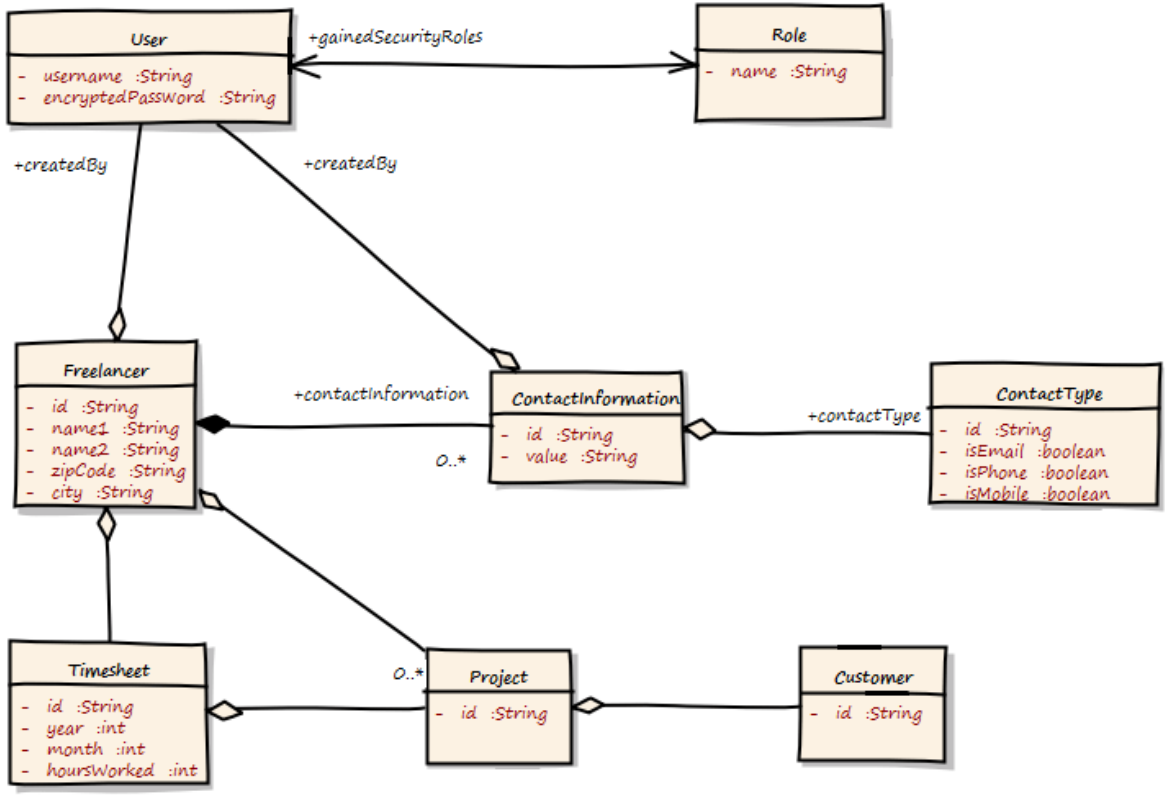
DDD - DOMAIN DRIVEN DESIGN EXERCISE

Requirements

- A company provides IT Body Leasing. They have some Employees, and a lot of Freelancers as Subcontractors. Currently, they use Excel sheets to manage their Customers, Freelancers, Timesheets and so on. The Excel solution does not scale well. It is not multi-user ready and does not provide security and audit logs. So, they decided to build a new web-based solution. Here are the core requirements:
  - A searchable catalogue of Freelancer must be provided
  - The new solution must allow to store the different Communication Channels available to contact a Freelancer
  - A searchable catalogue of Projects must be provided
  - A searchable catalogue of Customers must be provided
  - The Timesheets for the Freelancers under contract must be maintained.

First Solution

- Based on these requirements the development team decided to model everything using UML to get a big picture of the new solution:



First Solution - Problems

- The model is pretty straight forward
- There are Customers, Freelancers, Projects and Timesheets
- There is also a kind of user management to support role based security

- But wait, something is wrong here. There are some well hidden design flaws. Can you see them?

First Solution - Problems

- Here they are
  - It is a very big object graph. If they do not use Hibernate/ JPA lazy loading here, it will pretty sure run out of memory under heavy load
  - Why is the association between User and Role bi-directional?
  - The ContactType has some boolean flags to show what type it is: email, phone, mobile
  - The Freelancer class holds a list of Projects. This also means that Projects cannot be added without modifying the Freelancer object. This can cause transaction failure under heavy load, as possibly multiple users are adding Projects for the same Customer
  - What does ContactInformation mean? The requirements stated “Communication Channel”. Is it the same?
- The whole model seems to be more a kind of Entity-Relationship-Diagram instead of a software model
  - Also, where is the business logic? The team wanted to create some Business-Services around the model to store and retrieve data, the Entities are just POJOs managed by JPA.

Exercise 1

- Identify subdomains and create a Context Map (Strategic Design)

- Things to consider
  - One principle behind DDD is to bridge the gap between domain experts and developers by using the same language to create the same understanding
  - What is a Domain? A Domain is a “sphere of knowledge”, for instance the business the company runs. A Domain is also called a “problem space”, so the problem for which we have to design a solution.

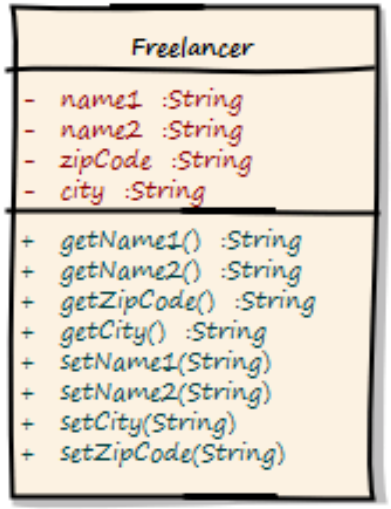
Exercise 2

- Create a domain model (Tactical Design)

- Things to consider
  - Change the classes to an appropriate type of the DDD Pattern Language (Entity, Aggregate, Value Object, …) → use the concept of stereotype here
  - Change the names of the classes to the names of your ubiquitous language
  - Try to arrange the elements to the found subdomains in the strategic design
  - Add a subdomain "common types" as "Shared Kernel".

Exercise 3

- Design behaviour for the Freelancer
  - Use Case: “Freelancer moved to new location”
  - Without DDD in mind, we could create a simple POJO as follows:



  - Create a DDD-solution. Things to consider
    - Where is the use case?
    - The setter might be called from other places. And implementing role-based security might become cumbersome, as we do not have the invocation context when a setter is invoked
    - Also, there is a missing concept in this model, the Address. It is modeled in a very implicit way, just by simple properties of the Freelancer class.

Literature

- This exercise is based on https://www.mirkosertic.de/blog/2013/04/domain-driven-design-example/