

ProPrä Praktikum

Module / Komponenten:

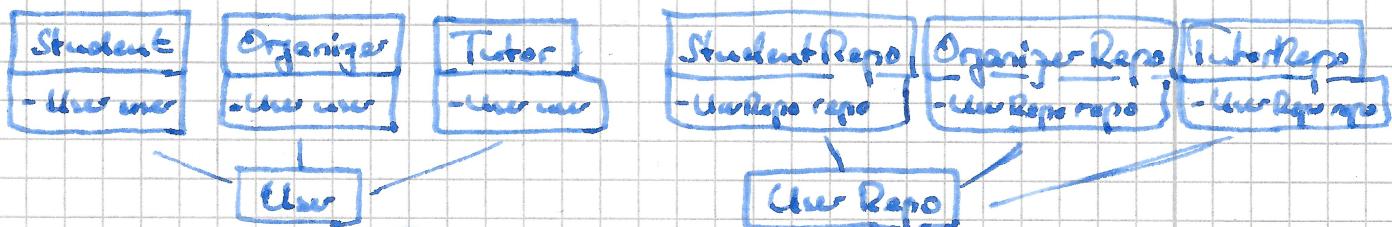
Wir modellieren User, Studenten, Organisatoren, Tutores nicht als Aggregates da sie von Natur aus Blutleer sind - sie speichern die gleichen Daten und werden von anderen Instanzen verwaltet, treten von sich aus aber kaum in Aktion. Sie sind POCOs. Sie sind auch nicht komplex, da sie nur Umstellung, GitHub-Hancks und E-Mail-Adressen tatsächlich haben. Das verleiht ihr Statur, sie nicht künstlich zu vollwertigen Aggregaten aufzupusten.

Sind die übergeordnete Abstraktion User zu modellieren macht weniger der Typsicherheit und Fachliche Sprache in der Business-Logik fürchterlich. Sie sollten daher getrennt als Tutor, Organisator, Student in ~~Form~~ klassen von separaten Klassen implementiert werden.

Ein Ziel war jedoch, dass alle Benutzer der Anwendung per Data JDBC in einer Tabelle gespeichert werden. Das ist mit ~~User~~ als ~~klassen~~ Oberklasse möglich, nicht aber mit drei getrennten Klassen.

Wir kommen nun Schuss, dass Vererbung hier ebenfalls ein Problem machen würde, da die Fallunterscheidung im Data JDBC Repository unzulänglich wäre. Wir entscheiden uns schließlich für Kapselung des Aggregaten Users durch fachliche Klassen Student, Tutor, Organisator per Komposition, und analog in den Repositories.

Wir haben ein Data JDBC - Repo für sog. "User", das aber nur als Backend für das fachlichen Repo dient.



Packaging: Wir legen jede Klasse mit ihrem Repository zusammen. So ist Typstabilität in Konstruktoren nicht notwendig, d.h. die Rolle von User muss nicht in Konstruktor von z.B. Student verarbeitet werden, da sich der Raum des StudentRepo bewahrt. Es handelt sich jedoch nicht um komplexe Aggregate, da wir User, bzw. UserRepo per Referenz von den anderen aus Wollen, nicht klopfer ID. Dann ist technisch notwendig. Wir holen aber das Beste aus allen Welten:

Pros:

- Am sieht der Geschäftslogik leichter wie
 - Semantik und Ubiquitous Language
 - Studenten, Organizer, Tutores als Aggregates (Verletzung nur dahinter)
 - Typsicherheit zur Komplexität (Student + Organizer, ...)
 - Keine Rollendeklaration nötig
- Im Backend können wir Data JDBC verwenden und haben nur eine Tabelle von Benutzern.
- Hohe Anpassbarkeit: Die Rollen können einzeln völlig frei modifiziert werden.

Cons:

- Viel Boilerplate-Code, zweimal drei fast gleiche Klassen
- Keine völlig saubere Aggregatstruktur