28.11.19 | SWT | ü8 (1)

Praktische Aufgaben

Tobias Ctistis, Gruppe 4

Aufgabe 1)

a) • Online-Seminarplatzwahl:

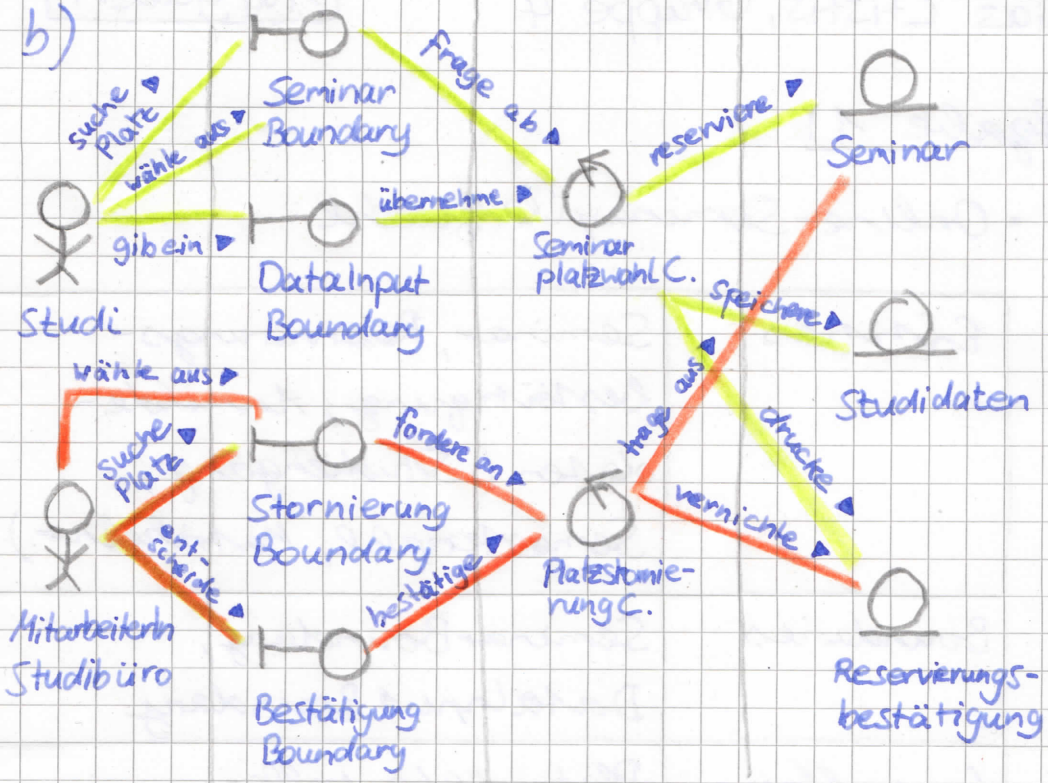
Entities	Seminar, Reservierungs-bestätigung, Anmelde-daten (Studiengang, Semesterzahl, Matrikelnr.)
Boundaries	SeminarBoundary, DataInputBoundary
Controller	PlatzwahlController
Beteiligte Akteure	Studi

• Platzstornierung:

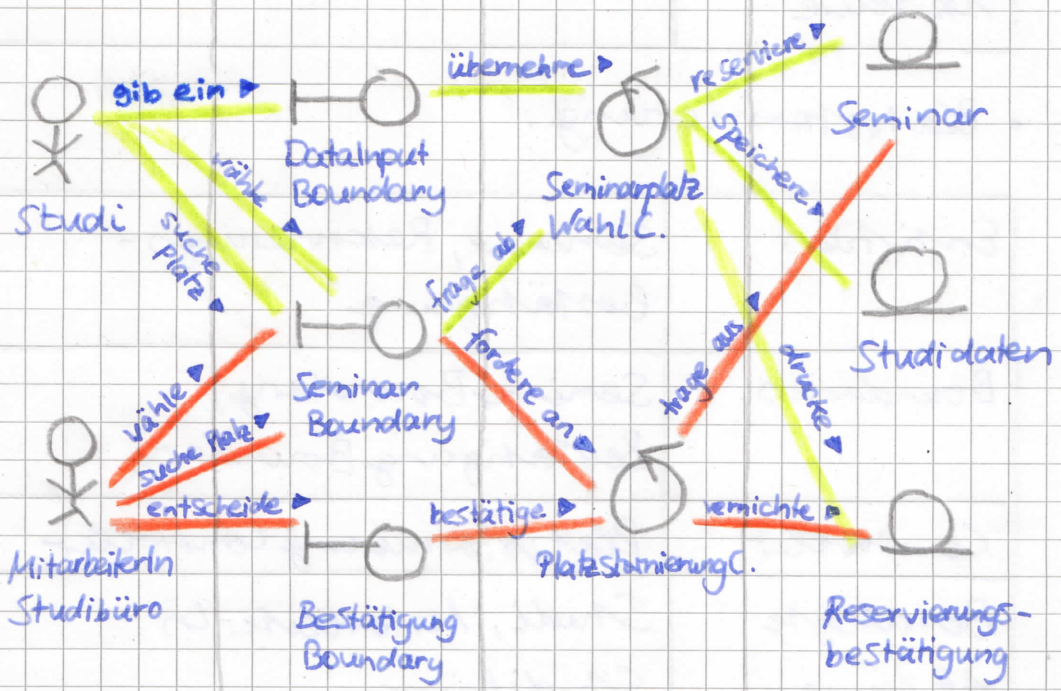
Entities	Seminar, Reservierungs-bestätigung
Boundaries	SeminarBoundary, BestätigungBoundary
Controller	PlatzstornierungController
Beteiligte Akteure	Studi, Mitarbeiter im Studibüro

28.11.19 | SWT | ü8 (2)

b)



c)

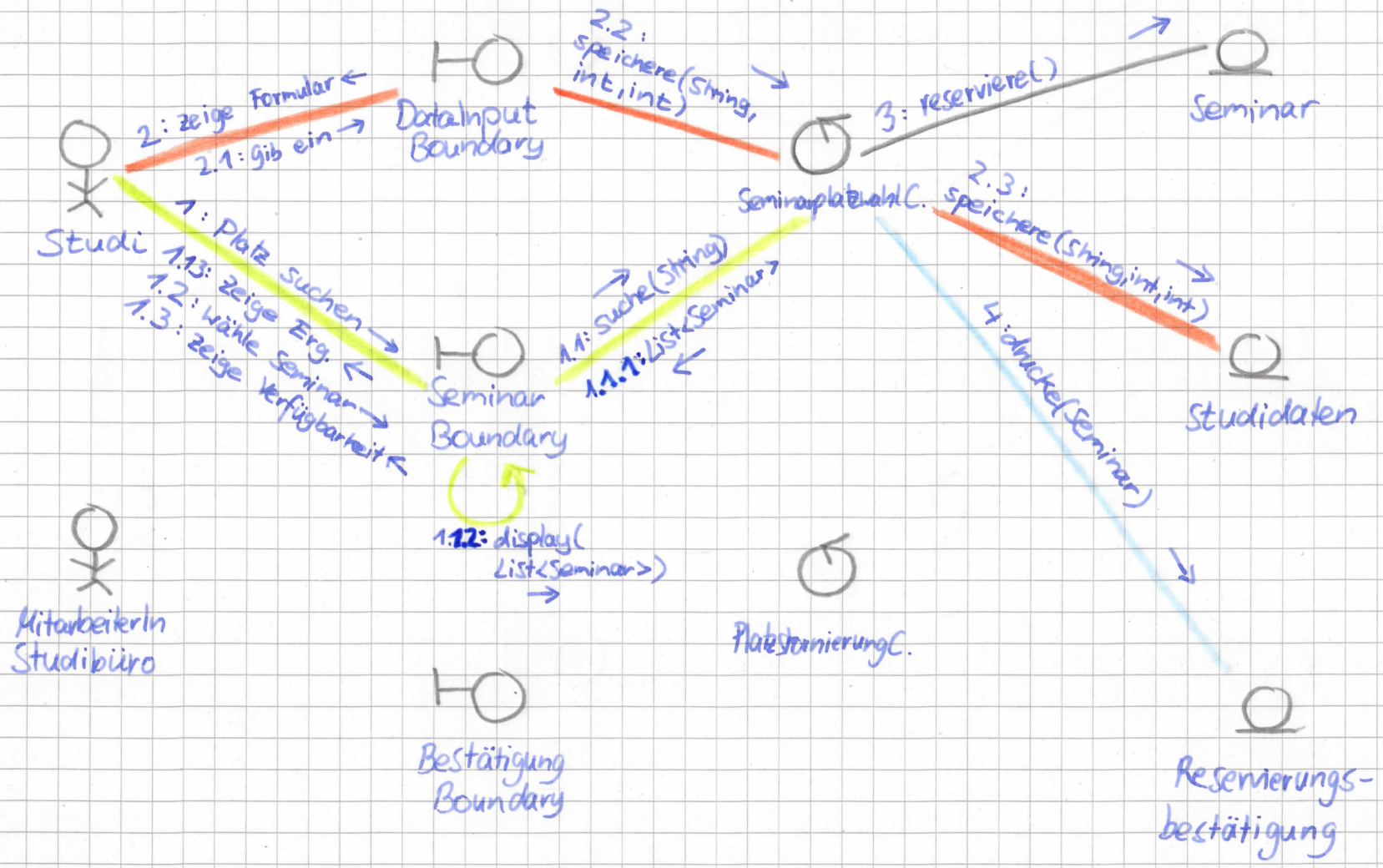


- SeminarBoundary, StornierungBoundary eigentlich identisch, je nach Akteur andere Funktionen

28.11.19 | SWT | Üb (3)

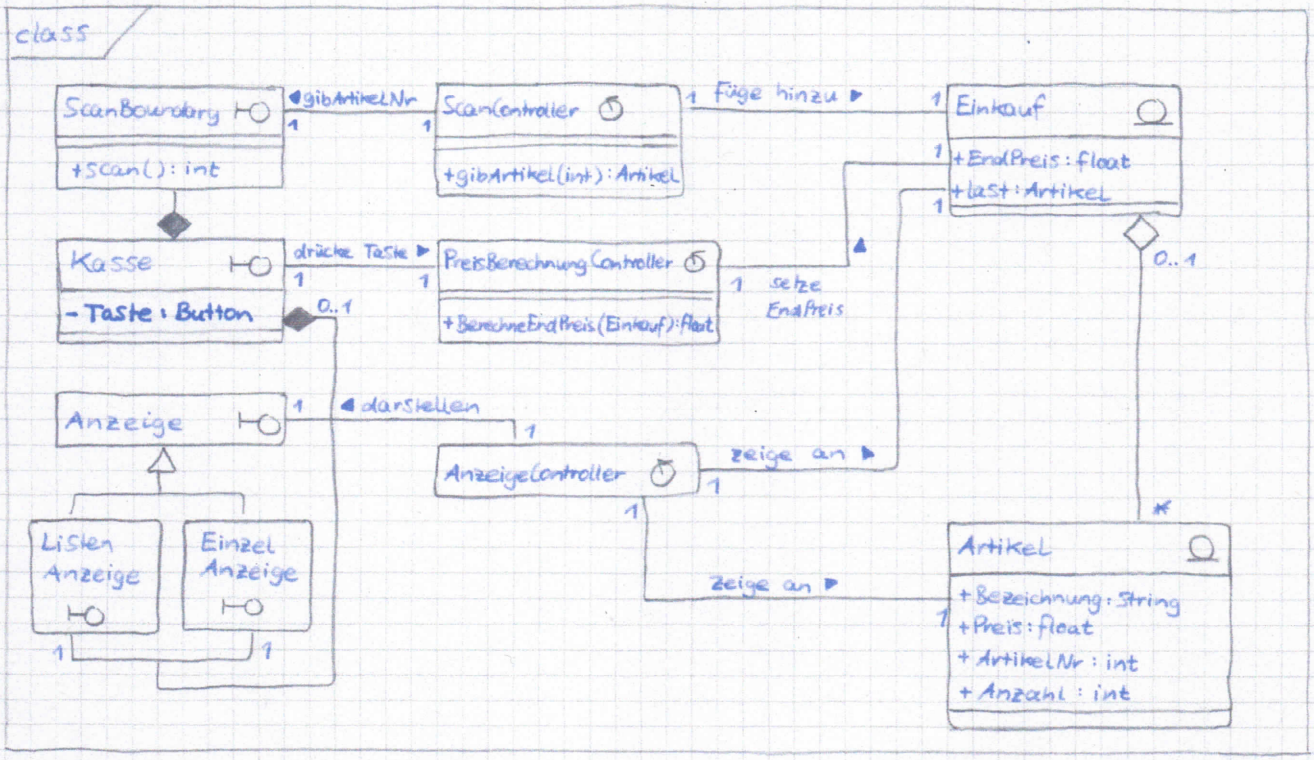
Tobias Chistis, Gruppe 4

Aufgabe 2)

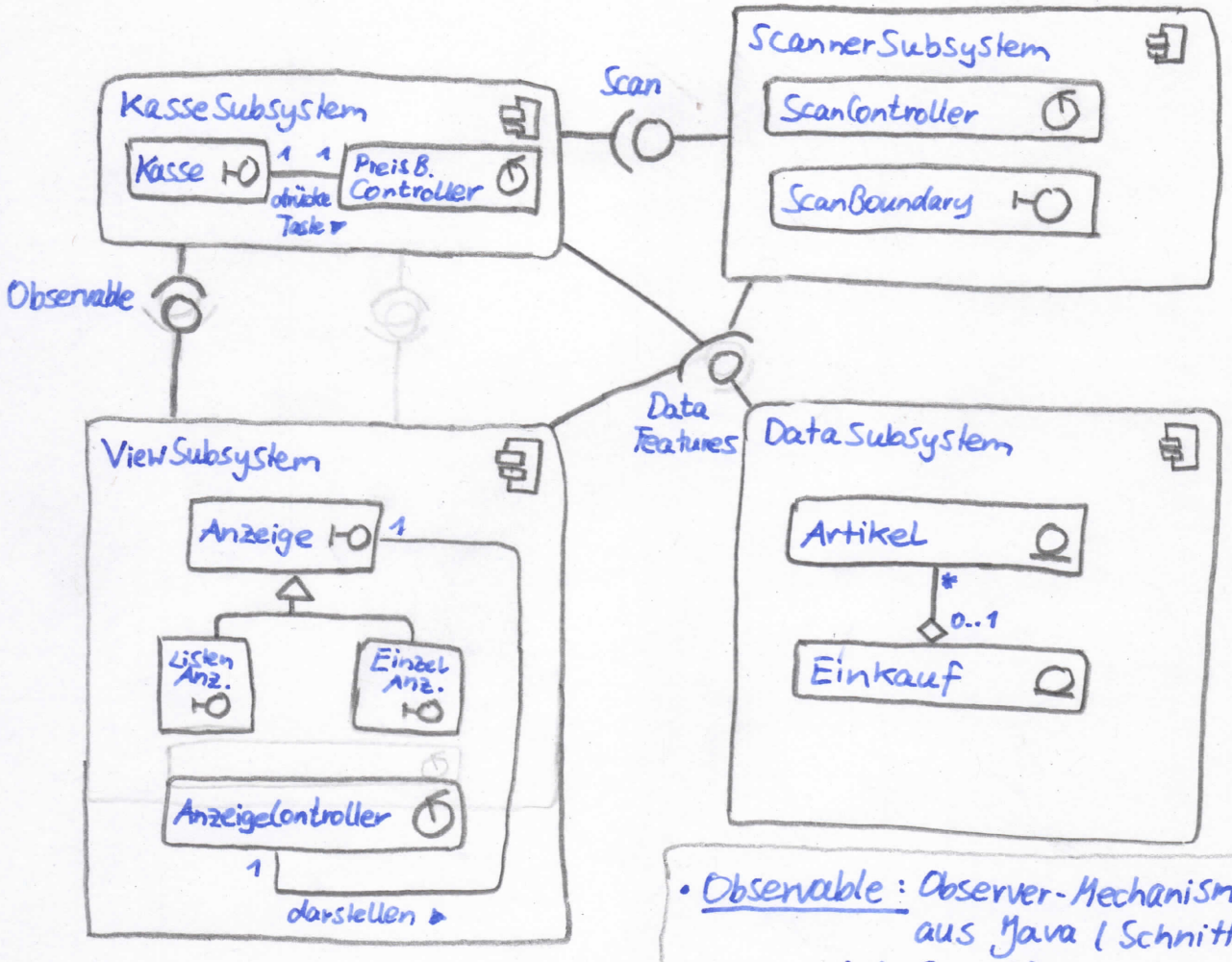


23.11.19 SWT Üb (7)

• Endgültiges, fertiges Modell:



Cmp • Methoden siehe oben



- Observable: Observer-Mechanismus aus Java (Schnittstelle)
- Scan: int Scan()
- Data Features: Datenoperationen auf Einkauf

SWT Ü8 27.11.19
Theoretische Aufgaben
==========

Aufgabe 3)

a) i) Komplexe Subsysteme durch einfache Schnittstelle nach außen abschirmen, um Komplexität zu reduzieren.

ii) Realisierung/Verfügbarmachen der angebotenen Dienste (und ihrer Schnittstellen) eines Subsystems nach außen

iii) Mit dem Singleton-Entwurfsmuster, um zentrale Kontrolle über die verwalteten Subsysteme zu haben

b) i) Es darf nur eine Instanz einer gegebenen Klasse erzeugt werden. Realisiert durch einen privaten Konstruktor und einer statischen getInstance-Methode, die die Instanz erzeugt oder, falls vorhanden, zurückgibt.

ii)
```
public class Singleton {

    private Singleton instance;

    private Singleton() {}

    public static Singleton getSingleton() {

        if (singleton == NULL) {
            singleton = new Singleton();
        } else {
            return singleton;
        }

    }

}
```

Variante: Instanz im Konstruktor erstellen.
Hier: In getInstance(), wenn Instanz das erste Mal benötigt.

Aufgabe 4)

a) **Pipes/Leitungen:** Leiten Datenstrom an eine/mehrere Filter weiter
Filter: Bearbeiten Datenstrom und leiten ihn an eine/mehrere Pipes weiter

Gute Methode, **große Datenströme (Messdaten etc.)** **parallelisiert** zu **filtern** (Filter können gleichzeitig arbeiten)

b) **Repository ist ein Datenpool**, in dem Clients Daten ändern/abfragen können. Clients kennen sich nicht gegenseitig, sondern nur ihr Repository. Sie registrieren sich beim Repo und werden von diesem bei Änderungen benachrichtigt. So werden Abhängigkeiten reduziert.

c)
- **2-Schichten-Client:**
  Ultra-Thin-Client z. B. Google Docs
  (Anwendungslogik/Präsentation/Datenhaltung
  im Web, eigener PC stellt HTML dar)
- **3-Schichten-Client-Server:**
  Client-Applikationsserver-Datenserver,
  Client lokal, andere beiden im Netzwerk
- **4-Schichten-Client-Server:**
  Wie 3-Schichten, nur weitere Aufteilung
  der Anwendungslogik (z. B. Webserver und
  Applikationsserver)
- **Peer-To-Peer:** PC-Fernsteuerungs-Software
  (Steuernder/Gesteuerter PC)

d)
- **Kohärenz:**
  Maß der Abhängigkeit innerhalb
  der Kapselungsgrenzen (der Subsysteme)
- **Kopplung:**
  Maß der Abhängigkeit zwischen
  den Kapselungsgrenzen (der Subsysteme)
- **Optimale Verteilung:**
  Maximale Kohärenz, minimale Kopplung

e)
- **Packages:**
  Namensräume, gliedern Klassen in Bereiche
  Zugriffe auf andere Packages jedoch oft ohne Beschränkung möglich!
- **Subsysteme:**
  Definieren Kapselungseinheiten
  Zugriff erfolgt von außen über
  festgelegte (schlanke) Schnittstellen

f)
- **Schicht (virtuelle Maschine):**
  Subsystem, welches Subsystemen höherer Ebenen
  Dienste anbietet, kennt höhere Schichten nicht,
  nur abhängig von Diensten tieferer Schichten
- **Partition:**
  Subsystem, welches Subsystemen auf
  gleicher Ebene Dienste anbietet.

g) **Middleware** ist eine Softwareschicht zwischen der Anwendung und dem Betriebssystem, die von diesem abstrahiert und bietet eine einheitliche Schnittstelle auf verschiedenen Betriebssystemen.

h) **Applikationsserver** bieten eine Laufzeitumgebung für Applikationen, die auf ihm laufen.
Er unterstützt als Middleware-Erweiterung auch Lebenszyklusmanagement, offene Datenquellen, etc.

i)
- **Client/Server:**
  Definiert klare Rollen der Beteiligten: Client oder Server
- **Peer-To-Peer:**
  Ermöglicht gleichzeitige, gleichberechtigte Kommunikation
  zwischen Beteiligten: Alle können mal Client, mal Server sein.