Beuth Hochschule für Technik Berlin, FB VI – Informatik und Medien
***Software Engineering (SE2)**, Hauptklausur SS 2013*
Termin: 5. Juli 2013 von 14^{15} bis 15^{45}

15 43 (3) 9

B

SOFTWARE ENGINEERING 2 — KLAUSUR SS 2013

Name, Vorname [redacted]

Matrikelnummer [redacted]

Pseudonym Prof. Snape
freiwillig, wenn Sie möchten, dass Ihr Klausurergebnis im Internet veröffentlicht wird

Platznummer 9.E
wird vom Betreuer (evtl.) zu Beginn der Klausur vergeben

Erster (X) **Zweiter** ( ) **Letzter** ( ) Versuch

Note für Punkte	0 bis 55,	ab 60,	ab 70,	ab 75,	ab 80,	ab 85,	ab 90,	ab 95,	ab 100,	ab 105,	ab 110
2,0 für 95 ?	**5,0**	**4,0**	**3,7**	**3,3**	**3,0**	**2,7**	**2,3**	**2,0**	**1,7**	**1,3**	**1,0**

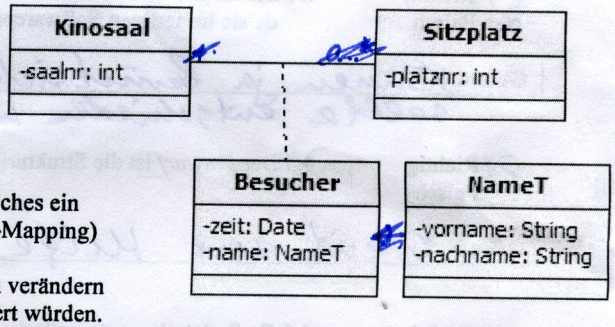
Lesen Sie zunächst alle Aufgaben sorgfältig durch. Sollten Sie Fragen haben, **können Sie diese in den ersten zehn Minuten laut stellen**. Spätere Fragen sind nicht mehr zulässig, denn laute Fragen stören, und leise Fragen widersprechen dem Gleichbehandlungsprinzip. Es sind keine Hilfsmittel zugelassen. Schreiben Sie Ihre Lösungen auf dieses Blatt (beachten Sie auch die Rückseite!), bzw. auf nummerierte leere Blätter mit Ihrem Namen; kennzeichnen Sie die Aufgabennummer eindeutig. Schreiben Sie am besten mit Kugelschreiber (Bleistift ist nicht zulässig!). Für falsche oder unverständliche Lösungen bekommen Sie grundsätzlich keine Punkte. Wenn aber aus Ihren Notizen oder Bemerkungen ersichtlich ist, dass Ihr Gedankengang korrekt war, können Sie Teilpunkte erreichen. Sie verlieren diese Möglichkeit jedoch, wenn Abschreiben oder Kommunikation während der Klausur nachgewiesen werden kann. Der Kern der Fragen wurde *kursiv* gesetzt. **Die Aufgaben sind ungefähr gleich aufwändig und jeweils 40 Punkte wert.**

Bearbeiten Sie bitte *unbedingt* die Aufgaben 1 und 4 (auf der Rückseite) und nur *eine* der beiden Aufgaben 2 und 3. Kennzeichnen Sie *deutlich*, welche Aufgabe Sie abgewählt haben.

1. AUFGABE — UNBEDINGT BEARBEITEN!

Alle Klassen des abgebildeten Klassendiagramms (aus der Entwurfsphase eines Softwareprojekts) sollen persistent sein.

1. Ergänzen Sie zunächst das Diagramm um *fachlich sinnvolle Multiplizitäten.*
2. *Notieren* Sie die Entitätenklassen in Java *und annotieren* Sie sie gemäß der von Ihnen gewählten Multiplizitäten. Hinweis: Es ist Ihnen gestattet, das Klassendiagramm zu modifizieren, sofern es dabei semantisch unverändert bleibt.
3. Modellieren Sie das Entity-Relationship-Modell (ER-Modell), welches ein Persistence Framework gemäß objekt-relationaler Abbildung (OR-Mapping) aus Ihren Entitätsklassen erzeugen würde.
4. Beschreiben Sie stichpunktartig, inwiefern sich das Materialisieren verändern würde, wenn statt der Getter-Methoden die Instanzattribute annotiert würden.



Lösung auf dem Extrablatt Nr. 1, 2

Punkte (0 5 10 15 20 25 (30) 35 40)

2. AUFGABE

Beschreiben Sie tabellarisch in Stichpunkten (kein Fließtext!)

1. vier Gründe, wann es sich *nicht empfiehlt*, das einzusetzende Datenbanksystem *selbst zu entwickeln*, sondern es sinnvoller ist, ein *Standardprodukt* zu wählen.
2. Erläutern Sie (ebenso stichpunktartig) die beiden Konzepte *Eager Creation* und *Lazy Creation* im *Singleton-Pattern.*
3. Nennen Sie *jeweils einen Vorteil* von Eager Creation und von Lazy Creation.

Lösung auf dem Extrablatt Nr.

Punkte (0 5 10 15 20 25 30 35 40)

Achtung, beachten Sie auch die Rückseite!

3. AUFGABE

Beschreiben Sie tabellarisch in Stichpunkten (kein Fließtext!)

1. vier *Anpassungen am Fachklassendiagramm* eines Analysemodells, um es in ein *Entwurfsmodell zu überführen.*
2. Erläutern Sie (ebenso stichpunktartig) die Funktion des *Observer-Patterns* (Beobachter-Muster).
3. Stellen Sie gemäß Observer-Pattern einen *Update-Vorgang* zwischen Model und View als *Sequenzdiagramm* dar.

Lösung auf dem Extrablatt Nr. 3

Punkte (0 5 10 15 20 25 30 35 (40))

4. AUFGABE

UNBEDINGT BEARBEITEN!

Kreuzen Sie die korrekte Antwort an und geben Sie Ihre *Begründung in Stichworten* dazu. Jede Einzelfrage ist fünf Punkte wert. Ohne Begründung wird Ihre Antwort nicht bewertet.

( ) Richtig
(X) Falsch

4.1 Das Unternehmen *PSI*, das uns vor zwei Wochen einen interessanten Fachvortrag gehalten hat, betätigt sich *inhaltlich mit der Entwicklung* von Software für die Forstwirtschaft.

Grund Das war aus der Metallindustrie! Ich war
Yeah!

(X) Richtig
( ) Falsch

4.2 *Assoziative Klassen* müssen im Entwurfsprozess *eliminiert werden.*

Grund Assoziative Klassen können auch werden nicht im Entwurfsprozess eingesetzt => Eliminiert

(X) Richtig
(X) Falsch

4.3 Das *Model-View-Controller-Konzept* (MVC) bezieht sich auf *alle drei Schichten* der im Entwurfsmodell eingesetzten Drei-Schichten-Architektur. nicht DB-Schicht

Grund Jedes Modell (Model, View, Controller) bzw. Schicht ist ein Teil des Konzepts und kann ohne einander nicht funktionieren.

( ) Richtig
(X) Falsch

4.4 Die *Kann-Kriterien* aus dem Pflichtenheft werden im Entwurfsmodell *nicht länger berücksichtigt*, da sie im fertigen Softwareprodukt ohnehin nicht realisiert sein werden. Geht auch später

Grund Können ja berücksichtigt werden, aber spätestens jetzt sollte entschieden werden, ob man die umsetzt!

(X) Richtig
( ) Falsch

4.5 Der *Entwurf* ist die Strukturierung des Softwaresystems *aus Anwendersicht.*

Grund Dient zur Hilfe bei der Implementierung.

(X) Richtig
( ) Falsch

4.6 Es findet eine automatische *Wertzuweisung* zwischen den Attributen der Backing Bean und den korrespondierenden Interaktionselementen der Benutzungsoberfläche statt.

Grund ~~Ach, keine Ahnung~~ !!

( ) Richtig
(X) Falsch

4.7 Ein *fachlich relevantes, eindeutiges* Attribut wie `kundenNr` oder `artikelId` ist als *Schlüsselattribut* (primary key) der Datenbanktabelle einer Entitätenklasse besonders geeignet.

Grund Ein generisches Schlüssel wäre besser geeignet.

(X) Richtig
(X) Falsch

4.8 Die *Fachlogikschicht* in der *Drei-Schichten-Architektur* darf auf die *darunter liegenden Schichten* zugreifen, jedoch nicht auf die *darüber liegenden.*

Grund ~~Kann~~ Nicht so richtig, da verschiedene Komponente mal auf untere, mal auf obere Schichten zugreifen.
Doch!

Punkte (0 5 10 15 20 (25) 30 35 40)

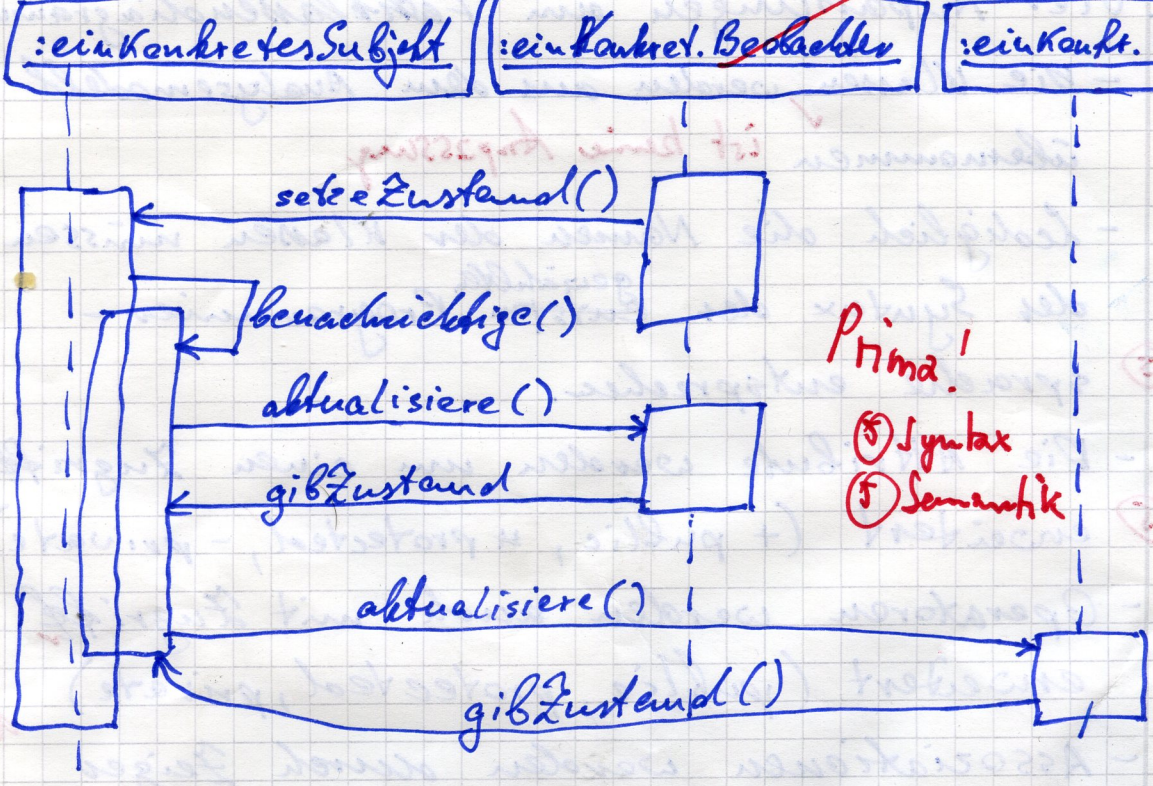
3.1. Vier Anpassungen am Faktklassendiagramm:

- Die Klassen werden aus dem Analysemodell übernommen ✓ ist keine Anpassung
- Lediglich die Namen der Klassen müssen der Syntax der ~~Entspr.~~ gewählten Programmiersprache entsprechen. (5)
- Die Attribute werden um einen Zugriff erweitert (+ public, # protected, - private) (5)
- Operatoren werden auch mit Zugriff erweitert (public, protected, private)
- Assoziationen werden durch Zeiger in uni- oder bidirektional dargestellt. (5) Navigation
- Bei n bzw m-Beziehung werden die Mengen der Assoziationen gespeichert
- Wenn möglich, sollten Klassen in Paketen gespeichert werden (4)

3.2. Observer-Pattern

- Ist ein objektorientiertes Verhaltensmuster
- Es handelt sich um eine Abhängigkeit zwischen einem Subjekt und seinen Beobachtern
- Das Hauptobjekt wird als Subjekt bezeichnet
- Es kann unbegrenzt viele Beobachter geben
- Ein Subjekt weiß nicht, wie viele Beobachter er hat und kennt die dementsprechend nicht
- Über eine Veränderung des oder beim Subjekt weiß der Beobachter erst, wenn dieser sich beim registrierten Objekt bzw. Subjekt informiert (5) Attach/Detach

rtsetzung ekseite!



Prima!
(5) Syntax
(5) Semantik

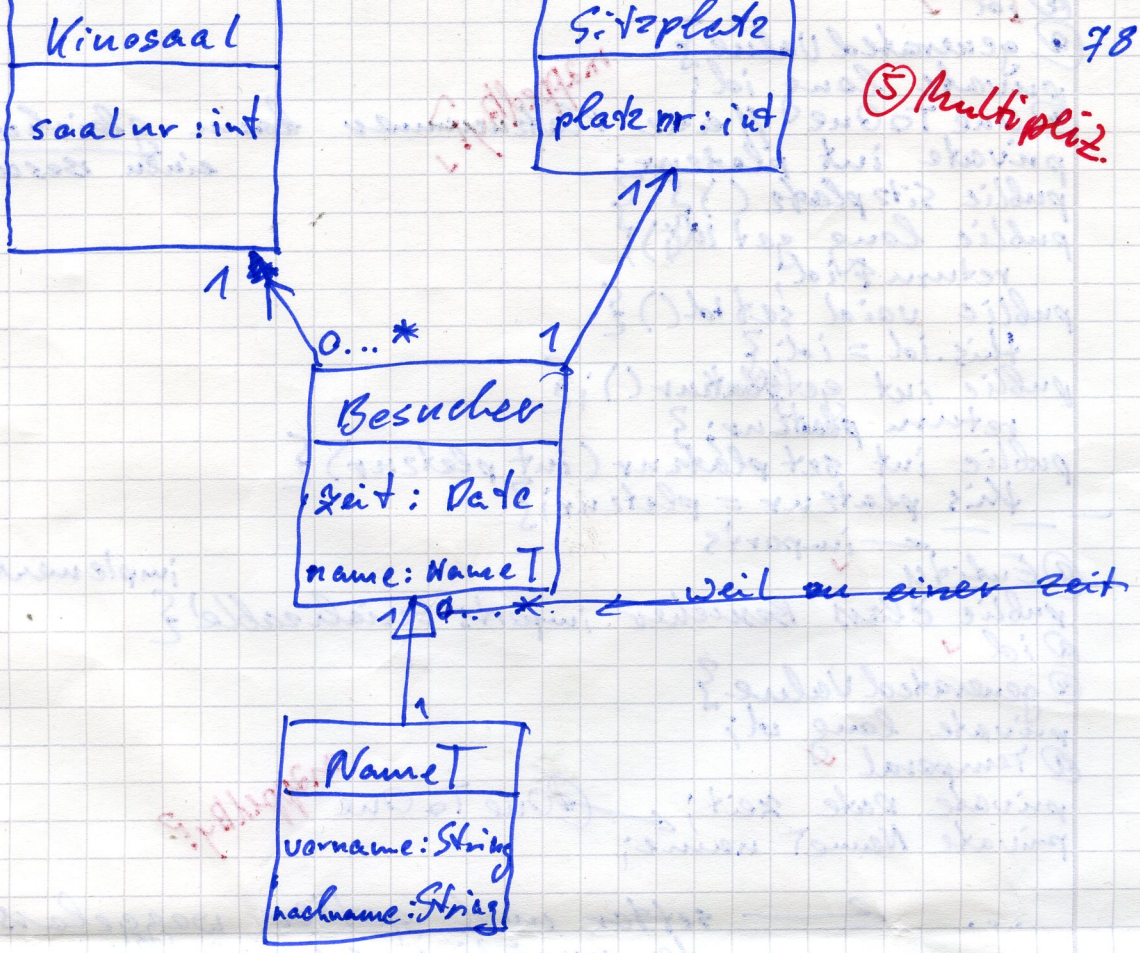
.2 Observer: Fortsetzung

- Durch ein Observer-Pattern wird eine enge Bindung vermieden, somit wird eine Wiederverwendbarkeit der eing. Komponente ga

Observer-Pattern wird angewendet, wenn:

- Eine Abstraktion 2 oder mehr Aspekte besitzt, die von einander abhängen
- Die Änderung eines Objekts die Änderungen anderer Objekte verbergt (5) Update
- Ein Objekt in der Lage sein sollte, andere Objekte zu benachrichtigen.

Schön!



786

⑤ Multiplizität

2.

```
import java.io.Serializable;
import javax.persistance.Entity;
@Entity ⑤
public class Kinosaal implements Serializable {
@Id ⑤
@generatedValue }
private long id
@ManyToOne ⑤ // Kinosaal kann viele Besucher haben
private int saalnr;
public Kinosaal () { }
public long getId(); {
    return id;}
public void setId (long id) {
    this.id = id;}
public int getsaalnr ();{
    return saalnr;}
public int setsaalnr (int saalnr){
    this.saalnr = saalnr;}
```

mapped by?

2 Rückseite

@Id ✓
@GeneratedValue}
private long id;
@OneToOne mappedBy? ✓ Platznummer kann gleichzeitig m
einen Besucher h
private int platznr;
public Sitzplatz() {}
public long getId();
return id;
public void setId() {
this.id = id;}
public int getPlatznr();{
return platznr;}
public int setPlatznr(int platznr) {
this.platznr = platznr;}

← imports

@Entity
public class Besucher imports Serializable implements Name {
@Id ✓
@GeneratedValue}
private long id;
@Temporal ✓
private Date zeit; ← @OneToOne mappedBy?
private NameT name;

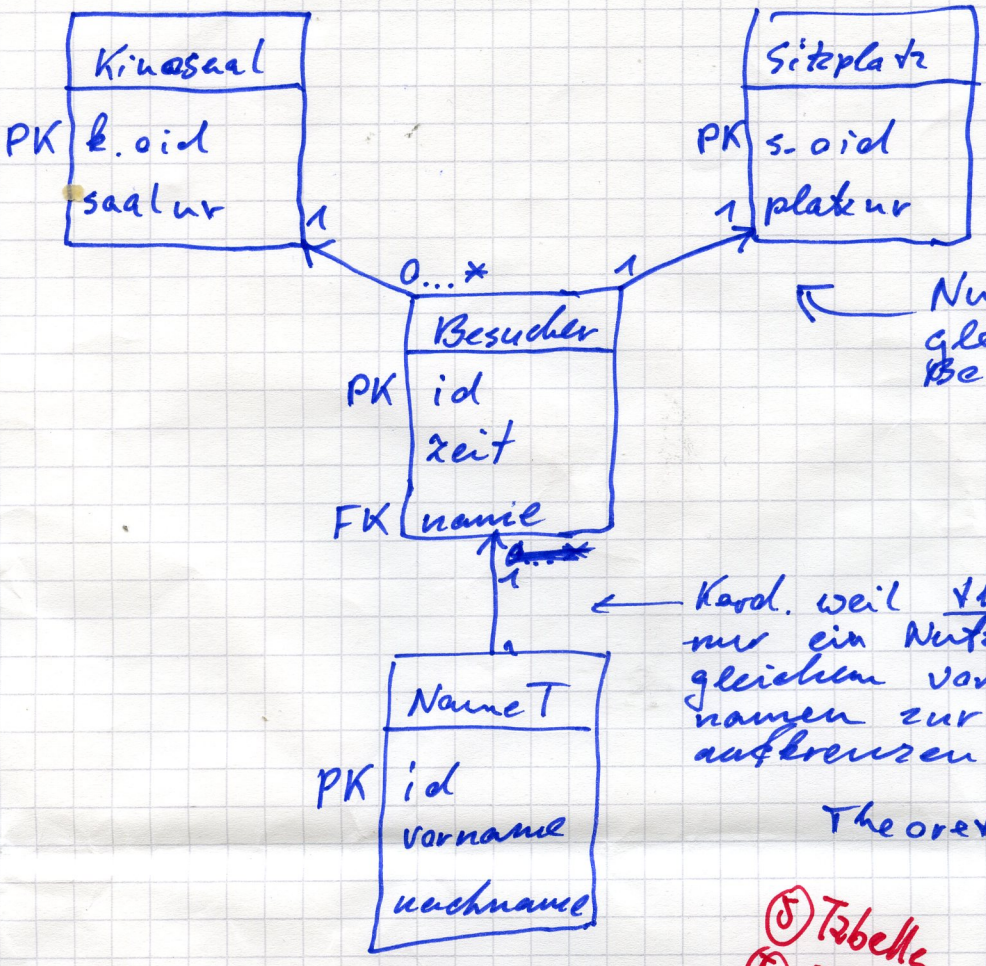
... ← setter und getter weggelassen, da unnötige Schreibarbeit...

import java.io.Serializable;
import javax.persistance.Entity;
@Entity ✓
public class NameT imports Serializable {
@Id ✓
@GeneratedValue}
private long id;
@OneToOne mappedBy? ✓
~~protected~~ ~~private~~ ~~vorname~~ String vorname; } protected weil privat nicht geht, da Besucher die NameT nutzt.
~~protected~~ ~~private~~ String nachname;

... } setter & getter-Methoden.

Irgendwo fehlt bei Ihnen ein „mappedBy", ohne das die Bidirektionalität

1.5. ERM



Aufgabenteil 1.4 fehlt

30