Beuth Hochschule für Technik Berlin, FB VI – Informatik und Medien
Software Engineering (SE2). Klausur WS 2010/11
Termin: 4. Februar 2011 von 12^{45} bis 13^{40}

13 30
3
7

SOFTWARE ENGINEERING 2 — KLAUSUR WS 2010/11

A

Name, Vorname _____

Matrikelnummer _____

Pseudonym asd
freiwillig, wenn Sie möchten, dass Ihr Klausurergebnis im Internet veröffentlicht wird

Platznummer _____
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
2,3 für 90 P.	5,0	4,0	3,7	3,3	3,0	2,7	2,3	2,0	1,7	1,3	1,0

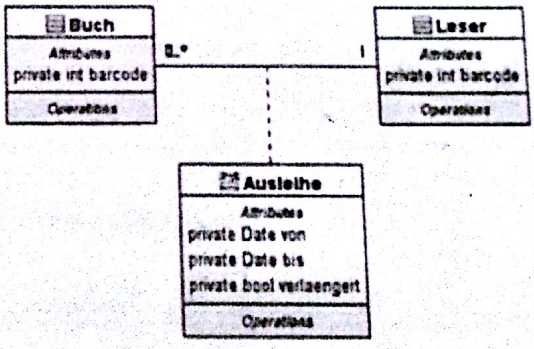
Lesen Sie zunächst alle Aufgaben sorgfältig durch. Sollten Sie Fragen haben, **können Sie diese in den ersten zehn Minuten laut stellen**. Spätere Fragen sind nicht mehr zulässig, denn laute Fragen stören, und leise Fragen widersprechen dem Gleichbehandlungsprinzip. Es sind keine Hilfsmittel zugelassen. 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 Aufgabe 4 (auf der Rückseite) und nur *zwei* der drei anderen Aufgaben. Kennzeichnen Sie *deutlich*, welche Aufgabe Sie abgewählt haben.

1. AUFGABE

Die drei Klassen des rechts abgebildeten Fachklassendiagramms *sollen persistent werden*:

1. *Schreiben und annotieren* Sie die Entitätenklassen in Java. Hinweis: Es ist Ihnen gestattet, das Klassendiagramm zu modifizieren, sofern es dabei semantisch unverändert bleibt.
2. *Modellieren Sie das Entity-Relationship-Modell*, welches ein Persistenz-Framework gemäß objekt-relationaler Abbildung (OR-Mapping) aus Ihren Entitätenklassen erzeugen würde.



Lösung auf dem Extrablatt Nr. 1

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

2. AUFGABE

1. Beschreiben Sie mindestens zwei übliche *Anpassungen und Erweiterungen am Fachklassendiagramm* eines Analysemodells, um dieses Diagramm in *ein Entwurfsmodell zu transformieren*.
2. Erläutern Sie die Funktion des *Observer-Patterns* (Beobachter-Muster) und stellen Sie einen *Update-Vorgang* zwischen Controller, Model und Views als *Sequenzdiagramm* dar.

Lösung auf dem Extrablatt Nr. 2-3

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

Achtung, beachten Sie auch die Rückseite!

3. AUFGABE

1. Erläutern Sie den *Unterschied* zwischen einem *Servlet* und einer *JSP-Seite*. Sie können dazu das rechts abgebildete Code-Fragment verwenden. Ist der Code *reentrant*?
2. Welche Aufgaben hat ein *Web-Container* (z.B. *Tomcat, Glassfish, JBoss*)? Beschreiben Sie kurz.

```
<body>
  <%! String time = new Date().toString(); %>
  Aktuelle Uhrzeit: <%= time %>
</body>
```

Lösung auf dem Extrablatt Nr.

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

4. AUFGABE

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.

(X) Richtig
( ) Falsch

4.1 Durch die Verwendung des Entwurfsmusters *Strategie* erhöht sich *die Anzahl der Objekte*. (0)

Grund die Verwendung von Strategieobjekten erhöht die Anzahl von Objekten in einer Anwendung das ist keine Begründung

( ) Richtig
(X) Falsch

4.2 *Verwaltungsmethoden* wie `get()` und `set()` sind im *Klassendiagramm eines Entwurfsmodells* nicht zu modellieren. (5)

Grund sinnvoll sind wie z.B. update() oder insert() zu modellieren

( ) Richtig
(X) Falsch

4.3 Ein *fachlich relevantes, eindeutiges* Attribut wie `kundenNr` oder `artikelId` ist als *Schlüsselattribut* (primary key) der Datenbanktabelle einer Entitätenklasse *besonders geeignet*. (5)

Grund ein generischer Schlüssel ist besser geeignet

( ) Richtig
(X) Falsch

4.4 Eine Klasse kann entweder *ganz oder gar nicht* persistiert werden. (5)

Grund einzelne Attribute können transient sein

(X) Richtig
( ) Falsch

4.5 Objektorientierte Vererbung lässt sich *auch auf relationale Datenmodelle* abbilden. (5)

Grund es gibt 3 Typen => Single, Table-per-class und Joined

(X) Richtig
( ) Falsch

4.6 *Data Binding* bezeichnet die *automatische Wertzuweisung* zwischen Beans-Attributen und UI-Elementen. (0)

Grund

( ) Richtig
(X) Falsch

4.7 Das Entwurfsmuster *Singleton* ist ein *klassenbasiertes Erzeugungsmuster*. (5)

Grund ist ein objektbasiertes Erzeugungsmuster

( ) Richtig
(X) Falsch

4.8 *Beans* haben stets *die Lebensdauer „request"*. (5)

Grund es gibt auch page, application und session

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

Aufgabe 1)

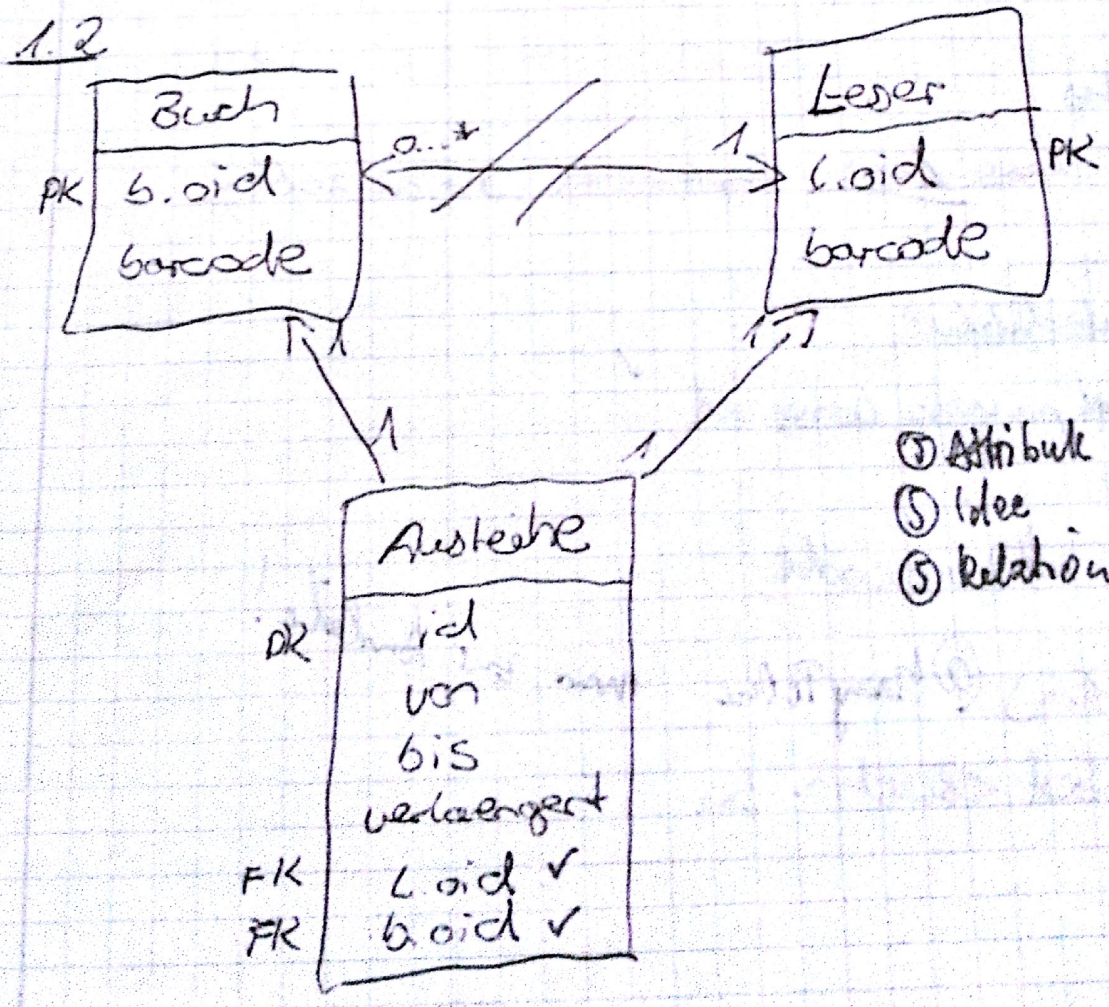
Wie haben Sie das Klassendiagramm modifiziert?

1.1

```
@Entity
public class Buch implements Serializable {
@Id
@GeneratedValue (s) Id
 private long id;

@Column
 private int barcode;

@ManyToOne OneToOne muss zu Anleihe!
 private Leser leser;

}
@Entity
public class Leser implements Serializable {
@Id
@GeneratedValue
 ~~private~~ private long id;
@Column
 private int barcode;
@OneToMany @ManyToOne muss zu Anleihe!
 private Set<Buch> bu;
}
```

```
public class Ausleihe implements Serializable{
@Id
@GeneratedValue ✓
 private long id;

@Column
 private Date von;
 private Date bis;
 private bool verlaengert;

@OneToOne ✓        assoziative Klasse muss aufgelöst werden!
 private Buch buch;      // weil je Buch ein Ausleih
 private Leser leser;    // gibt sowie je Leser
                         // deshalb hab ich mich für den
                         // OneToOne entschieden
}
```

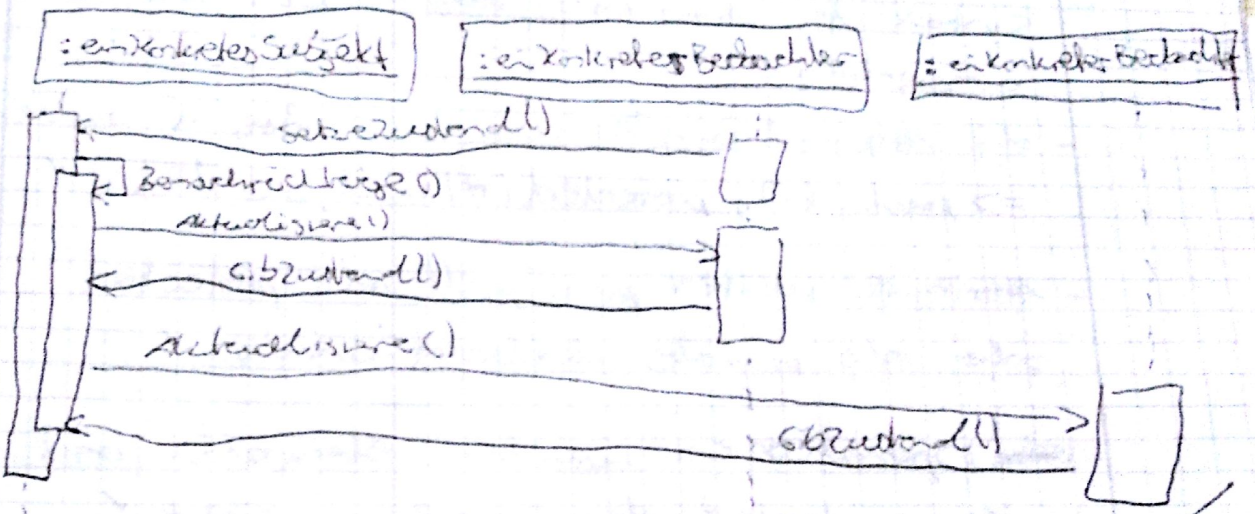
1.2



① Attribut
② Idee
③ Relationen

Aufgabe 2)

2.2



10

Observer-Pattern:

- ist ein objektbasiertes Verhaltensmuster ✓
- es handelt sich um eine Abhängigkeit zwischen dem Subjekt und ~~seinen~~ seinen Beobachter. ✓
- das Hauptobjekt wird als Subjekt bezeichnet ✓
- es kann unbegrenzt viele Beobachter haben ✓
- Subjekt kennt seine Beobachter nicht ✓
- über eine Änderung am Subjekt werden Beobachter, erst wenn sie sich beim Subjekt registrieren, informiert ✓
- durch das Observer-Pattern wird eine enge Bindung vermieden ✓
  ↳ eine Wiederverwendbarkeit der einzelnen Komponenten ✓

Prima!

10

8.1

- die ~~Klassen~~ Klassen werden aus dem Analysemodell übernommen ✓
- lediglich die Namen der Klassen müssen nun der Syntax der verwendeten Programmiersprache entsprechen ✓
- die Attribute werden um einen Zugriff erweitert => public (+), protected (#) und private (-) ✓
- Attribute sollten prinzipiell als protected oder als private deklariert werden ✓
- ~~Obs~~ Operationen werden ebenfalls mit public, protected oder private erweitert ✓
- Assoziationen werden durch Zeiger in uni- oder bidirektional dargestellt ✓
- Bei einer 1 bzw. n-Beziehung werden Klassen von Assoziationen gespeichert ✓
- Aggregation wird übernommen
  ↳ jedoch muss das Ganze stets seine Teile kennen
- die Komposition wird ebenfalls übernommen
- es muss sich auf eine Sprache und eine objektorientierte Programmiersprache geeinigt werden ✓
- wenn möglich sollten Klassen in Pakete zusammengefasst werden z.B. ui, dg, db ✓

Amen! (20)

zu Aufgabe 2.2)

Beobachter-Muster wird angewandt, wenn ...
... eine Abstraktion 2 oder mehr Aspekte besitzt, die voneinander abhängen ✓
... die Änderung eines Objekts die Änderung anderer Objekte verlangt ✓
... ein Objekt in der Lage sein sollte, andere Objekte zu benachrichtigen ✓

10