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
1,7 für 100	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 2 und 4 (auf der Rückseite) und nur *eine* der beiden Aufgaben 1 und 3. Kennzeichnen Sie *deutlich*, welche Aufgabe Sie abgewählt haben.

1. AUFGABE

1. Beschreiben Sie tabellarisch in Stichpunkten den Begriff *Objektpersistenz* und grenzen Sie ihn vom *direkten Datenbankzugriff* mittels SQL-Anweisungen ab.
2. Erläutern Sie im Kontext JPA die Begriffe *Entity Manager*, *Persistence Context* und *Cache*. Erläutern Sie zwei *Transaktionszustände* im Cache.
3. Skizzieren Sie grob die Klassenstruktur eines *Virtual Proxy-Musters* und erläutern Sie daran, wie mit dessen Hilfe ein *„teurer" Datenbankzugriff* vorübergehend *vermieden* werden kann.

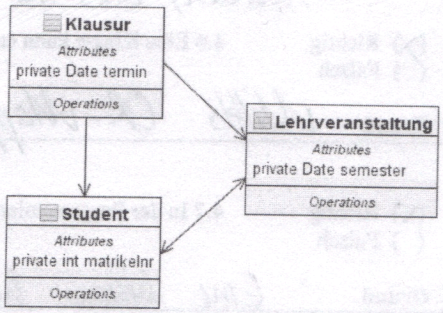
Lösung auf dem Extrablatt Nr.

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

2. AUFGABE — UNBEDINGT BEARBEITEN!

Alle drei Klassen des abgebildeten Fachklassendiagramms aus einem Entwurfsmodell sollen persistent sein.

1. Ergänzen Sie das Klassendiagramm zunächst um *fachlich sinnvolle Multiplizitäten.*
2. Notieren Sie die Entitätenklassen in Java und annotieren Sie sie gemäß der von Ihnen gewählten Multiplizitäten. „Student" ist hierbei die führende Klasse („owning side").
3. Modellieren Sie das Entity-Relationship-Modell (ER-Modell), welches ein Persistence Framework gemäß objekt-relationaler Abbildung (OR-Mapping) aus Ihren Entitätenklassen erzeugen würde. Machen Sie deutlich, wo sich darin die *Fremdschlüssel befinden.*



Lösung auf dem Extrablatt Nr. 4-6

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

Achtung, beachten Sie auch die Rückseite!

3. AUFGABE

Notieren Sie stichpunktartig (kein Fließtext!):

1. Beschreiben Sie mit vier Beispielen, in welchen Situationen es sich *nicht empfiehlt*, das einzusetzende Datenbanksystem *selbst zu entwickeln*, sondern es sinnvoller ist, ein *Standardprodukt* zu wählen.
2. Erläutern Sie das Konzept der *Lazy Creation* im *Singleton*-Entwurfsmuster, und vergleichen Sie es mit der *Lazy Materialization* mithilfe des *Proxy*-Musters in einem *Persistenz-Framework*.

Lösung auf dem Extrablatt Nr. 1-3

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.

(X) Richtig ( ) Falsch — 4.1 Durch die Verwendung des Entwurfsmusters *Strategie* erhöht sich *die Anzahl der Objekte*.

Grund Wenn viele verschiedene ~~Klas~~ Eigenschaften dargestellt werden sollen, es geht aber um Objekte (0)
erhöht sich die Klassenzahl, dafür jede E. eine eigene Klasse erstellt werden muss

(X) Richtig ( ) Falsch — 4.2 *Verwaltungsmethoden* wie `get()` und `set()` sind im *Klassendiagramm eines Entwurfsmodells* zu modellieren.

Grund Auch weil Attribute privat gesetzt werden sollen. Entwurf ist das Spiegelbild des späteren Programms! (5)

( ) Richtig (X) Falsch — 4.3 Die *Kann-Kriterien* aus dem Pflichtenheft werden im Entwurfsmodell *nicht länger berücksichtigt*.

Grund Werden genauso berücksichtigt. Es muss auch nicht entschieden werden (0)
ob sie verwaltet werden sollen.

(X) Richtig ( ) Falsch — 4.4 Objektorientierte Vererbung lässt sich *nicht auf relationale Datenmodelle* abbilden.

Grund Auf jeden Fall nicht so wie es in der OO passiert. Es gibt aber natürlich (5)
Möglichkeiten es doch irgendwie abzubilden mit @Inheritance z.B.

(X) Richtig ( ) Falsch — 4.5 Ein Web-Server funktioniert nach dem *HTTP-Request-Response-Paradigma*. Das klassische *Model-View-Controller-Konzept* lässt sich hier nicht anwenden.

Grund Beim MVC-Konzept wäre es auch möglich nur eine „response" zu (5)
senden, das ist mit HTTP nicht möglich

(X) Richtig ( ) Falsch — 4.6 Eine Klasse kann entweder *ganz oder gar nicht* persistiert sein.

Grund Mittels OR-Mapping wird ~~er~~ Persistenz erreicht. Begründung unpassend (0)

(X) Richtig ( ) Falsch — 4.7 In der Programmiersprache *Java* gibt es *keine Mehrfachvererbung*.

Grund Eine Klasse kann nur von einer Klasse erben. ~~Man kann~~ (5)
Man kann aber beliebig viele Interfaces einbauen.

(X) Richtig ( ) Falsch — 4.8 Im *Klassendiagramm des Entwurfs* sollten möglichst viele *Assoziationen unidirektional* sein.

Grund bidirektionale Assoziationen können das Programm (5)
fehleranfälliger machen

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

3. Aufgabe

1.) – viele Nutzer greifen zeitgleich auf Datenbank zu (5)
– große Menge an Daten ist zu verarbeiten (5)
– das Auftraggebende Unternehmen benutzt schon seit Jahren ein bestimmtes DBSystem,
→ Umsteigen ist teuer (5)
– Benutzer sollen ad-hoc-Abfragen selbständig durchführen (5)

Wenn man selbst ein DBSystem schreibt, muss natürlich die Stabilität des Systems gewährleistet sein. Greift man auf eine Standardlösung zurück, ist in der Regel sichergestellt, dass sie stabil läuft und im Fehlerfall das System wiederhergestellt werden kann und keine Daten gelöscht werden sind. Außerdem verfügen Standardlösungen über einen Support. Es gibt also wenige Gründe warum man selbst ein DB-System programmieren sollte. Dies führt zu unnötigem Wartungsaufwand und Fehlerquellen. ✓

2.) Lazy Creation (Singleton)

- beim Singletonpattern wird sichergestellt, dass es nur eine Instanz des Objekts gibt. ✓
- Lazy Creation bedeutet, dass einem einzige Objekt erst instanziiert wird, wenn es tatsächlich gebraucht wird. (5)
- dies spart Zeit beim Zugriff und ~~es entfällt~~ es entfällt Synchronisationsarbeit (5)

Lazy Materialization (Proxy)

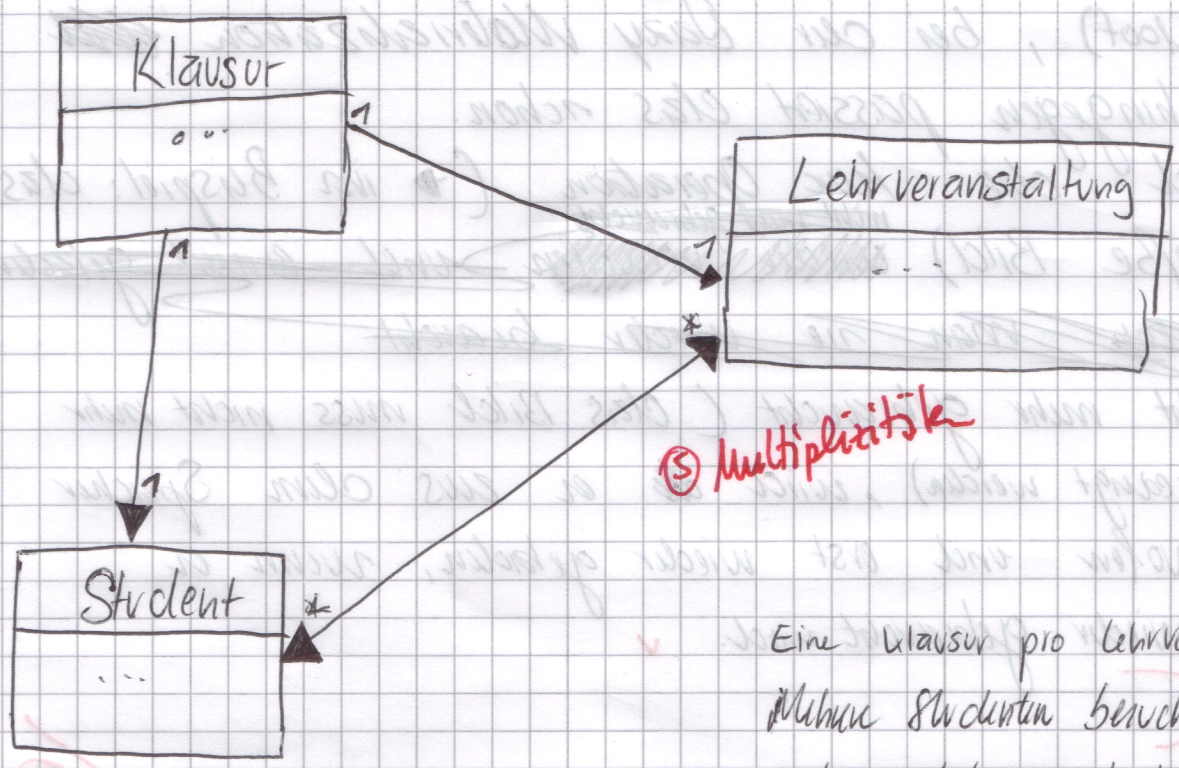
- ein Proxy wird in einem Persistenz-Framework vorgeschaltet, um die Performance von teuren Operationen (z.B. große Bilddaten) zu verbessern ✓ (3)
- Lazy ~~Materialization~~ Materialization bedeutet, dass teure Operationen erst dann ausgeführt werden, wenn sie wirklich gebraucht werden. (4)
  Die große Bilddatei wird z.B. erst angezeigt, wenn der Nutzer sie auch tatsächlich betrachtet.

Lazy Creation v. Lazy Materialization sind ähnliche Konzepte.
Bei der Lazy Creation aber ~~ist~~ wird die erzeugte

Instanz der Objekts nicht mehr verwofen, wenn sie gerade nicht mehr gebraucht wird (wäre ja auch doof), bei der Lazy Materialization ~~[illegible]~~ dahingegen passiert das schon.
Wird die ~~[illegible]~~ Operation (→ im Beispiel: das große Bild) ~~nicht mehr gebraucht, [illegible] verworfen und [illegible] gezeichnet, [illegible] wenn sie wieder [illegible]~~ nicht mehr gebraucht (das Bild muss nicht mehr angezeigt werden), wird es aus dem Speicher gelöscht und erst wieder geladen, wenn es wieder gebraucht wird. ✓

40

2. Aufgabe

① 

⑤ Multiplizitäten

Eine Klausur pro Lehrveranstaltung
Mehrere Studenten besuchen
mehrere Lehrveranstaltungen
Eine Klausur für einen Studenten

②

```
@Entity
public class Student extends Serializable {
⑤ @Id @GeneratedValue private long id;
   private Lehrveranstaltung lv;
   private Klausur k;
   private int matrikelnummer;
⑤ @ManyToMany
   private Lehrveranstaltung getLehrveranstaltung() {
        return lv;
   }
   @ManyToOne
   private Klausur getKlausur() {
        return k;
   }
   ...
}
```

④

@Entity
public class Utilisateur extends Serializable {
@ID @GeneratedValue private long id;
private Student st;

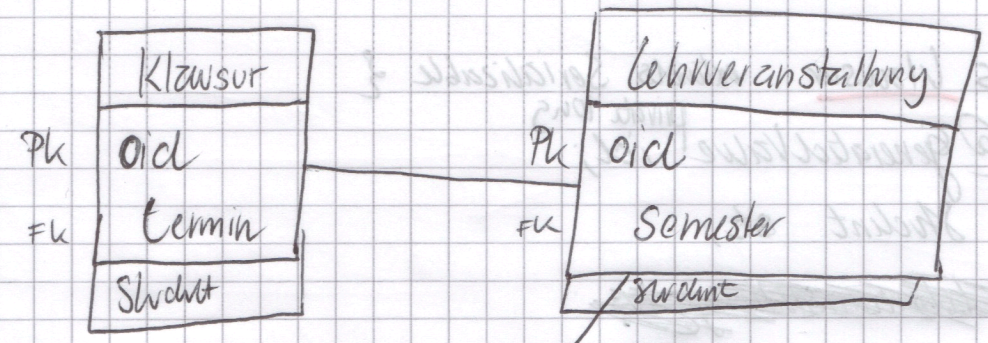
@OneToOne (mappedBy="u")
private Student getStudent() {
return student;
}

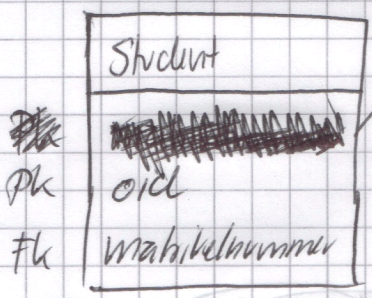
...

}

@Entity
public class Universitaire extends Serializable {
@ID @GeneratedValue private long id;
private Student st;
@ManyToMany (mappedBy="universitaire")
private Student getStudent() {
return student;
}
...
}

(3)





▷ Join Table erforderlich!

▷ Student ist fühlende Seite!

⑤ Stundenplan
- Tabellen gefunden
- Fk ersetzen

⑤

(30)