13 30 (7)

Berliner Hochschule für Technik, FB VI – Informatik und Medien
***Software Engineering (SE2)**, Hauptklausur SoSe 2024*
Termin: 19. Juli 2024 von 12^{15} bis 13^{45}

SOFTWARE ENGINEERING 2 — KLAUSUR SoSe 2024

A

Name, Vorname [redacted]

Matrikelnummer [redacted]

Pseudonym [redacted]
freiwillig, wenn Sie möchten, dass Ihr Klausurergebnis im Internet veröffentlicht wird

Platznummer [redacted]
wird vom Betreuer (evtl.) zu Beginn der Klausur vergeben

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

20 63
25 bis 58
Sorry!

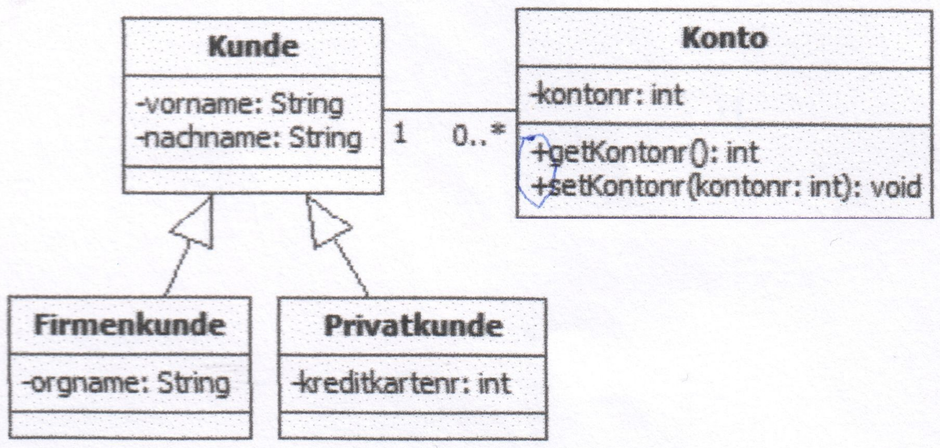
Note für Punkte	0 bis 29,	ab 30,	ab 35,	ab 40,	ab 45,	ab 50,	ab 55,	ab 60,	ab 65,	ab 70,	ab 75
	5,0	**4,0**	**3,7**	**3,3**	**3,0**	**2,7**	**2,3**	**2,0**	**1,7**	**1,3**	**1,0**

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 höchstens *zwei einseitig* beschriebene DIN A4-Zusatzblätter** 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 1 und nur *eine* der beiden Aufgaben 2 und 3. Kennzeichnen Sie *deutlich*, welche Aufgabe Sie abgewählt haben. Geben Sie zum Aufgabenblatt höchstens *ZWEI EINSEITIG* BESCHRIEBENE DIN A4-Zusatzblätter ab. Weitere Blätter und andere Formate werden nicht bewertet.

1. AUFGABE — UNBEDINGT BEARBEITEN!

Die abgebildeten Klassen sollen *persistent werden*:



1. *Notieren* Sie die Entitätenklassen in Java *und annotieren* Sie sie gemäß der bereits vorgegebenen Multiplizitäten und Navigationsrichtungen. Überlegen Sie genau, an welchem Objektattribut (oder welcher Getter-Methode) das *mappedBy*-Attribut am sinnvollsten ist.
2. Verwenden Sie die Vererbungsstrategie *JOINED*.
3. Begründen Sie stichpunktartig, weshalb das *mappedBy-Attribut* in der von Ihnen hierfür gewählten Klasse *besonders sinnvoll* ist.
4. Modellieren Sie das *Entity-Relationship-Modell*, welches ein Persistenz-Framework gemäß objekt-relationaler Abbildung aus Ihren Entitätenklassen generieren würde. Machen Sie darin besonders deutlich, wo sich die *Primärschlüssel* (primary keys), *Fremdschlüssel* (foreign keys) und weitere *Attribute befinden*.
5. Beschreiben Sie stichpunktartig, worin sich der Vorgang des Materialisierens und Dematerialisierens der Objekte unterscheiden würde, wenn *statt der Getter-Methoden* die *Objektattribute* annotiert würden. Denken Sie in diesem Zusammenhang auch an den Mechanismus *Java Reflections*.

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

2. AUFGABE

Beschreiben Sie stichpunktartig (kein Fließtext!):

1. „GoF"-Muster werden gemäß ihrer *Aufgabe kategorisiert*. Welche Aufgabenkategorien kennen Sie? Beschreiben Sie sie.
2. Welche drei Merkmale machen eine Klasse zu einem *Singleton-Muster*?
3. Erklären Sie die *Aufgaben* der im Observer-Pattern verwendeten Operationen.
4. Was sind die *Gemeinsamkeiten* und die Unterschiede der beiden Entwurfsmuster *Fabrikmethode* und *Strategie*?

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

3. AUFGABE

1. *Beschreiben* (nicht nur nennen!) Sie drei übliche *Anpassungen oder Erweiterungen am Fachklassendiagramm* eines Analysemodells, um dieses in *ein Entwurfsmodell zu transformieren.* Wählen Sie hierzu bitte nicht die „Anpassung der Bezeichner an die ausgewählte Programmiersprache“, denn das wäre zu einfach. ;)[redacted]
2. Was ist ein *abgeleitetes Attribut* und was eine *abgeleitete Assoziation*? Und was haben beide gemeinsam?
3. Warum sollten *bidirektionale Assoziationen* im Klassendiagramm des Entwurfsmodells vermieden werden?
4. Beschreiben Sie mindestens zwei Vorteile, die sich aus der Verwendung der Drei-Schichten-Architektur ergeben.

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

Die folgende Fläche dürfen Sie gern für Ihre eigenen Notizen verwenden. Sie geht NICHT in die Bewertung ein.

Berliner Hochschule für Technik, FB VI – Informatik und Medien
***Software Engineering (SE2)**, Hauptklausur SoSe 2024*
Termin: 19. Juli 2024 von 12^{15} bis 13^{45}

Bewertungsbogen der SE2-Klausur im Sommersemester 2024

Name, Vorname

Persistenzaufgabe „Kunde/Konto“ (unbedingt zu bearbeiten)

(5) Annotation `@OneToMany` in Klasse *Kunde*, sowie `@ManyToOne` in Klasse *Konto*
(5) Annotation `@OneToMany(mappedBy="kunde")` in Klasse *Kunde*, Attribut in *Konto*
(5) Annotation `@Inheritance(strategy=...JOINED)` in Basisklasse *Kunde*
(5) Begründung, warum *mappedBy* in Klasse *Kunde* besonders sinnvoll ist (o.ä.)
(5) Alle Tabellen *Konto*, *Kunde*, *Firmenkunde* und *Privatkunde* (ohne Join Tables) gefunden
(5) Attribute in den richtigen Tabellen eingetragen (*vorname*, *nachname*, *orgname*, ...)
(3) Fremdschlüssel *kunde* in Tabelle *Konto* eingetragen, Fremdschlüssel auf die Basisklasse
(5) Sinnvolle Erklärung für *Objektattribut vs. Getter-Methoden* gegeben

---

30 Punkte von 40 (max. 8 x 5)

Entwurfsmuster

( ) *Strukturmuster* beschrieben (*Strukturen* sind Assoziationen oder Vererbungen)
( ) *Verhaltensmuster* beschrieben (Verhalten wird durch *Methoden* definiert)
( ) *Erzeugungsmuster* beschrieben (Objekte werden durch das Muster *instanziiert*)
( ) Merkmale des Singletons gut erläutert (Klassenattr., stat. `getInstance()`, priv. Konstrukt.)
( ) `attach()` und `detach()` im Model skizziert (Liste zum Eintragen der Views)
( ) `notify()` im Model und Empfang von `update()` im View bzw. Controller skizziert
( ) Fabrikmethode *und* Strategie produzieren/liefern ein Objekt *einer ihrer* Unterklassen (o.ä)
( ) Fabrikmethode bestimmt *allein*, welcher Objekttyp tatsächlich produziert wird (o.ä.)

_____ Punkte von 40 (max. 8 x 5)

Erläuterungen „Entwurfsmodell“

(–) Erste Anpassung oder Erweiterung am Fachklassendiagramm in Ordnung
(5) Zweite Anpassung oder Erweiterung am Fachklassendiagramm in Ordnung
(5) Dritte Anpassung oder Erweiterung am Fachklassendiagramm in Ordnung
(–) Bedeutung von *abgeleitetem Attribut* und *abgeleiteter Assoziation* gut erläutert
(–) Gemeinsamkeit(en) von *abgeleitetem Attribut* und *abgeleiteter Assoziation* gut erläutert
(5) *Bidirektionale Assoziationen* vermeiden (z.B. Join Tables, Fehlerpotenzial, Synchronisation)
(5) Erster Vorteil aus der Verwendung der Drei-Schichten-Architektur (z.B. Austauschbarkeit)
(5) Zweiter Vorteil aus der Verwendung der Drei-Schichten-Architektur (z.B. Wartbarkeit)

---

25 Punkte von 40 (max. 8 x 5)

SE2 Klausur 19.07.21

1. Aufgabe

1./2.

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Kunde {

    @Id
    @GeneratedValue
    private int k_id;
    private String vorname;
    private String nachname;

    @OneToMany(mappedBy = "kunde")
    private List<Konto> konten;

    @OneToMany(mappedBy = "kunde")
    public List<Konto> getKonten(){
        return this.konten;
    }
}

@Entity
public class Konto {

    @Id
    private int kontonr;
    @NotNull private Kunde kunde;

    public int getKontonr() {
        return this.kontonr;
    }

    public void setKontonr(int kontonr){
        this.kontonr = kontonr;
    }

    @ManyToOne
    public Kunde getKunde() {
        return this.kunde;
    }
}
```

~~@Entity~~

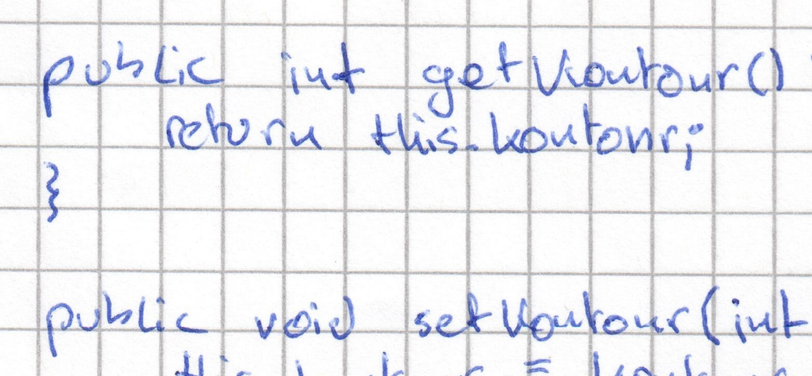
```
@Entity
public class Firmenkunde extends Kunde {

    ~~private String orgname;~~
    @Id
    @GeneratedValue
    private int f_id;
    private String orgname;
}
```

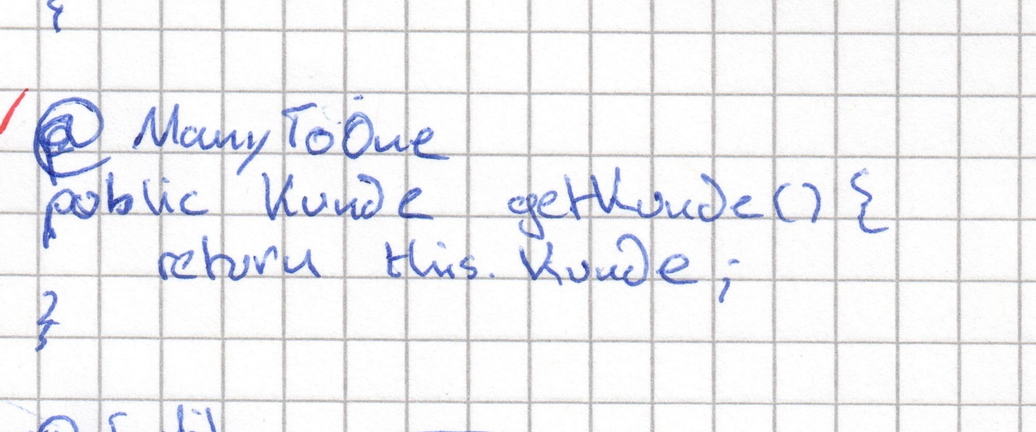
```
@Entity
public class Privatkunde extends Kunde{

    @Id
    @GeneratedValue
    private int p_id;
    private int kreditkartennr;
}
```

3.
- mappedBy-Attribut dort besonders sinnvoll weil in Richtung "toOne"
- umgekehrten müsste man einen Join Table erstellen, weil Tabelle als Wert gegen 1. NF. verstößt :)

5.
- was annotiert wird wird bewertet
- wenn getter annotiert ↳ werden diese vom Persistence-Framework (FW) genutzt um an Attributwerte zu kommen
- wenn Objektattribute annotiert ↳ Persistence-FW nutzt Java Reflections um Attributwerte zu lesen/setzen

4.





- FK auf Tabelle Kunde!

SE 2 Klausur 19.07.24

3. Aufgabe

1.
- Es müssen die für die Persistierung üblichen Klassen ergänzt werden wie

  PersistenceBag
  Proxy
  DBBroker, RelationalBroker etc.

  und vermutlich keine DB benötigt wird?
- Es sollten eventuell dabei Entwurfsmuster umgesetzt werden, (siehe auch erster Punkt), so aber auch i.d.R. Singletons, Fabrikmethoden, die bekannte Konzepte in vorherigen Diagrammen bringen. ✓
- Es müssen ~~Assoz~~ Assoziative Klassen ~~für~~ in echte Klassen mit Assoziationen übersetzt werden, da diese nicht programmiert werden können, wobei natürlich die Assoziationen alle überarbeitet werden sollten (bidirektionale wenn möglich auflösen).

2. ↳ (quasi eigener Punkt) ✓

Och! Schade! :(

3.
- durch bidirektionale Assoziationen → Objekte der beiden Klassen in anderer auffindbar sein für Persistierung ✓
  ↳ bei unidirektional nicht
- wenn bidirektional und bspw. ManyTo1 / To1Many / ManyToMany muss gemappt oder bei ManyToMany sogar ein Join Table erstellt werden → hoher Aufwand / Speicherplatz ✓

4.
- Durch Schichten wird SRP - Single-Responsibility-Principle umgesetzt, sodass Klassen nur eine Zuständigkeit haben und leichter getrennt werden können.
- Außerdem können durch die Trennung der View (Darstellungs-schicht) leichter Änderungen an der Fachlogik vorgenommen werden, ohne die darauf die meistens vorkommenden oder Daten zu verändern. ✓

:)

↑ müssen