3 13 36 (3)

Beuth Hochschule für Technik Berlin, FB VI – Informatik und Medien
***Software Engineering (SE2)**, Klausur WS 2011/12*
Termin: 3. Februar 2012 von 12[15] bis 13[45]

SOFTWARE ENGINEERING 2 KLAUSUR WS 2011/12

B

Name, Vorname [redacted]

Matrikelnummer

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

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

Erster ( ) **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
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 Aufgabe 4 (auf der Rückseite) und nur *zwei* der drei anderen Aufgaben. Kennzeichnen Sie *deutlich*, welche Aufgabe Sie abgewählt haben.

1. AUFGABE

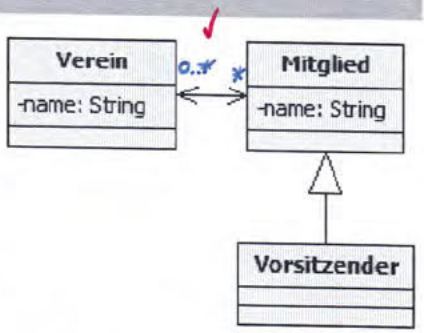
Beschreiben Sie tabellarisch in Stichpunkten:
1. Welche Aufgaben hat ein *Web-Container* (beispielsweise Tomcat, Glassfish). ✓
2. Was bedeutet „Deployment", und welche Angaben enthält ein *Deployment-Deskriptor*? ✓
3. Was versteht man unter *clientseitiger Validierung* und *serverseitiger Validierung*? Für *welche Aufgaben* ist die clientseitige Validierung zuständig und für welche die serverseitige? ✓

Lösung auf dem Extrablatt Nr. 1

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

2. AUFGABE

1. Alle Klassen des rechts abgebildeten Klassendiagramms sollen persistent sein. Ergänzen Sie das Diagramm um *fachlich sinnvolle Multiplizitäten*.
2. Schreiben Sie den erforderlichen *Java-Quelltext* auf und *annotieren* Sie ihn gemäß der von Ihnen gewählten Multiplizitäten und der bereits vorgegebenen Navigationsrichtungen. Verwenden Sie die Vererbungsstrategie *SINGLE_TABLE*.
3. Modellieren Sie das Entity-Relationship-Modell, welches ein Persistence Framework gemäß objekt-relationaler Abbildung (OR-Mapping) aus Ihren Entitätsklassen generieren würde. Hinweis: Machen Sie in Ihrem ER-Modell genau erkenntlich, wo sich Primär- und Fremdschlüssel befinden.



Lösung auf dem Extrablatt Nr. 2

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

3. AUFGABE

Erläutern Sie tabellarisch in Stichpunkten den *Zusammenhang* zwischen einer JSP-Seite und einem *Servlet*. Sie können dazu das rechts abgebildete Code-Fragment verwenden. Ist der Code reentrant?

```
<body>
  <%! String time = new Date().toString(); %>
  Die aktuelle Uhrzeit ist <%= time %>
</body>
```

Lösung auf dem Extrablatt Nr.

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

4. AUFGABE (Richtig-/Falsch-Fragen)

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 *Beans* haben stets *die Lebensdauer „request"*.

Grund Sie können auch Sessionscoped, ~~oder~~ Sitescoped und an die Lebensdauer der Webanwendung gebunden sein. (5)

(X) Richtig
( ) Falsch
4.2 Das Entwurfsmuster *Singleton* ist ein *objektbasiertes Erzeugungsmuster.*

Grund Es kann dadurch nur eine Instanz des Objektes geben. (5)
(Object.getInstanzC();)

(X) Richtig
( ) Falsch
4.3 Ein *fachlich relevantes, eindeutiges* Attribut wie `kundenNr` oder `artikelId` ist als *Schlüsselattribut* der Datenbanktabelle einer Entitätsklasse *nicht geeignet.*

Grund Da sich die Fachlichkeit ändern kann und die ID nichts mit der Fachlogik verbunden werden soll. (5)

(X) Richtig
( ) Falsch
4.4 Ein *Phase Listener* ermöglicht es, die Berechtigung des Benutzers beim Zugriff auf eine Webseite vom *Web Framework* „JavaServer Faces" überprüfen zu lassen.

Grund Der Phasen listener kann in die einzelnen Phasen der JSF eingebunden werden (after Phase, befor Phase) um zu prüfen ob ein Benutzer angemeldet ist oder nicht. (5)

(X) Richtig
( ) Falsch
4.5 *Assoziative Klassen* müssen im Entwurfsprozess *transformiert werden.*

doch! gerade das gelt!

Grund Assoziative Klassen können nicht ORM in einer Datenbank gespeichert werden und müsse in Skalare zerlegt werden oder in eine Klasse überführt werden. (0)

(X) Richtig
( ) Falsch
4.6 Das *Model-View-Controller-Konzept* (MVC) bezieht sich auf *alle drei Schichten* der im Entwurfsmodell eingesetzten Drei-Schichten-Architektur.

Grund MVC besteht aus Benutzerschicht (View), Fachschicht (Model) und Persistenzschicht, ~~welche~~ wird auf die mit einem Controlle/Service zugegriffen (0)
die DB-Schicht nimmt nicht teil

( ) Richtig
(X) Falsch
4.7 Die *Kann-Kriterien* aus dem Pflichtenheft werden im Entwurfsmodell *nicht länger berücksichtigt*, da sie im fertigen Softwareprodukt mit hoher Wahrscheinlichkeit nicht realisiert sein werden.

Grund Die Kann Kriterien werden genau wie die Muss-Kriterien im Entwurfsmodell berücksichtig. (5)

(X) Richtig
( ) Falsch
4.8 In einer Klasse ist es auch möglich, *nur einzelne Attribute* zu persistieren.

@Transient (5)

Grund Attribute die nicht persistent sein sollen mit @Transit versehen.

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

[redacted]

1. Aufgabe:

1.
- Webserver (Glassfish, Tomcat) stellen den Webcontainer bereit mit den in der Java EE 6 Spezifikation beschriebenen Schnittstellen ✓
- Webcontainer verwaltet die einzelnen Bean (POJO's) (5)
- Webserver nimmt Request entgegen und leitet sie an den Webcontainer weiter ✓
- Webcontainer bearbeitet Request und baut entsprechend der Anfrage die Seite zusammen und übergibt sie dem Webserver als HTML, dieser sendet diese als respons an den Client zurück (5)

2. - Deployment bedeutet veröffentlichen ✓
- es werden die Java.java Dateien compiliert in Java.Class Dateien und im Webcontainer bereitgestellt (5)
- Der Deployment-Deskriptor (DD) beschreibt auf welche Anfragen eines Clients reagiert wird, hier sind zum Beispiel aufgelistet index.jsp, index.html und index.xhtml (5)
- dabei wird im DD fest gelegt auf welche URL reagiert wird ✓

  zum Beisbiel */faces im DD besagt das alle Anfragen die /faces in der URL haben. durch das JSF Framwork bearbeitet werden ✓
- Deployment: Anwendung wird mit der entsprechenden Ordnerstruktur auf dem Webserver veröffentlich und als Anwendung.war Datei zum hochladen auf einen Webserver bereitgestellt ✓

3.

- Clientseitige Validierung bedeutet das alle Benutzereingaben in der View (Browser) auf korrekte Eingabe und gültige Werte geprüft werden (5)
- Serverseitige Validierung bedeutet das die Eingegebenen Werte die mit submit Button ~~an de~~ vom Benutzer an den Server gesendet wurden, vor der Verarbeitung auf dem Server nochmals auf gültige Werte geprüft werden. (5)

(5)

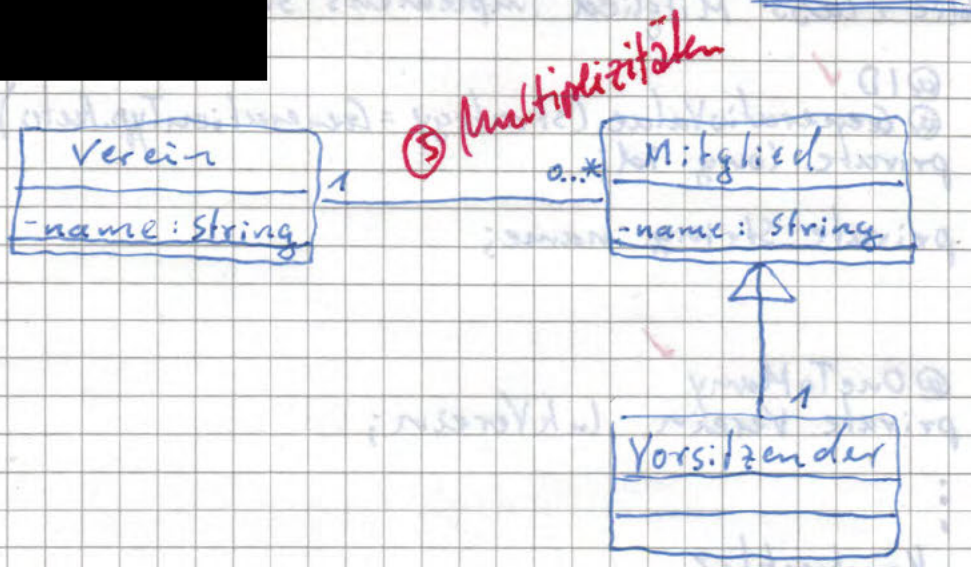
- Clientseitige Validierung soll dem Benutzer helfen Formulare richtig aus zu füllen. Pflichtfelder, Datumsangaben etc. = Syntax
- Serverseitige Validierung soll die Sicherheit erhöhen um Datenbank injektion zu verhindern und das einschleusen von Schadcode vermeiden. = Semantik

35

[redacted]

Blatt 2

1.



⑤ Multiplizitäten

@Entity ⑤
public class Verein implements Serializable {

@ID ⑤
@GeneratioValue (strategy = GenerationTyp.Auto)
private Long id;

private String name;

⑤ ⑤
@OneToMany (mappedby = lnkVerein)
private List<Mitglied> lnkMitglied;

: Konstruktor

:
public String getName() {
return this.name;
}
public void setName (String name) {
this.name = name;
}
public List<Mitglied> getLnkMitglied () {
return this.lnkMitglied;
}
public void setLnkMitglied (List<Mitglied> lnkMitglied) {
this.lnkMitglied = lnkMitglied;
}
public Long getId () {
return this.id;
}
public void setID (Long id) {
this.id = id;
}
:
equals ();
hashCode ();
toString ();
}

```
@Entity ✓
public class Mitglied implements Serializable{

    @ID ✓
    @GeneratioValue (strategy = GenerationTyp.Auto)
    private long id;

    private String name;


    @OneToMany ✓
    private Verein lnkVerein;

    :
    :
     Konstruktor
    :
    public long getId () {
      return this.id;
    }
    public void setId (long id){
      this.id = id;
    }
    public String getName () {
       return this.name;
    }
    public void setName (String name) {
       this.name = name;
    }
    public Verein getVerein () {
       return this.lnkVerein;
    }
    public void setVerein (Verein verein) {
       this.lnkVerein = verein;
    }
    :
       equels ();
       hashcode ();
       toString ();
}
```

[redacted] @Inheritance ⑤

@[redacted](inheritance = SINGLE_TABLE) ▷ Diskriminator?

@Entity

public class Vorsitzender extends Mitglied implements Serializable {

@Id ✓

@GeneratedValue(strategy = GenerationTyp.Auto)

private long id;

:

Konstruktor

:

@[redacted]

public long getId() {
  return this.id;
}

public void setId(long id) {
  this.id = id;
}

: equals();

hashcode();

toString();

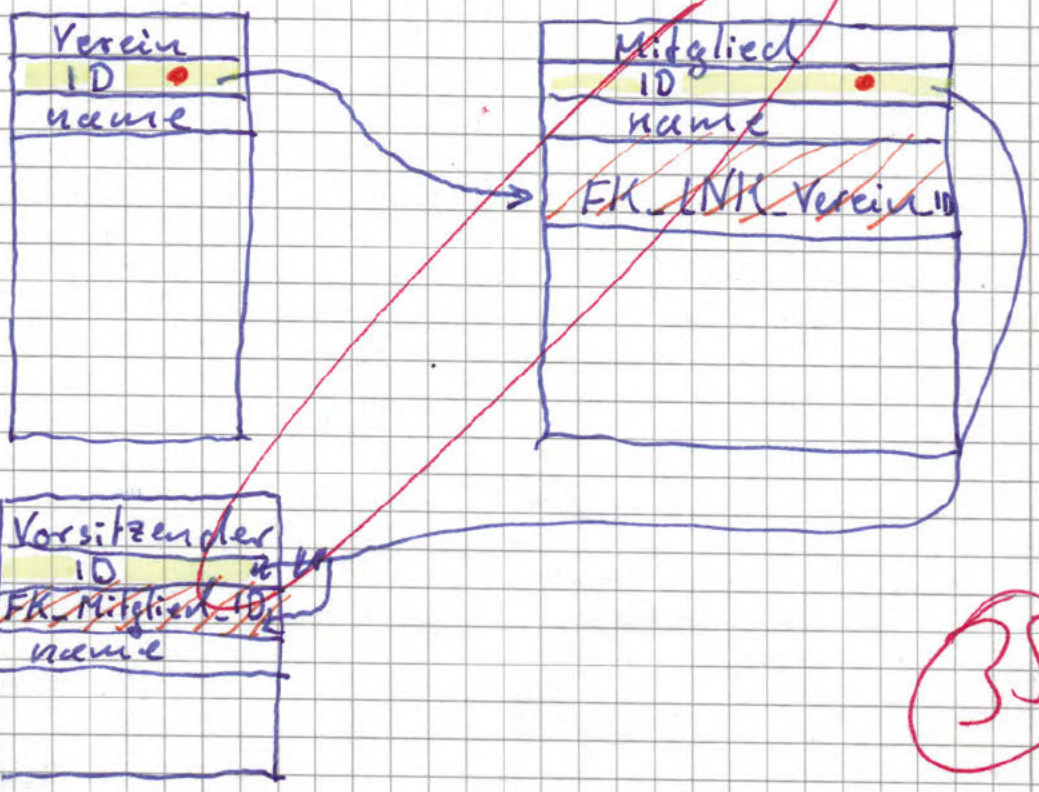
}

3.

⑤ key

▷ Diskriminator?

SingleTable!!!



35