Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Herbst 2018

66116

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Datenbanksysteme, Softwaretechnologie**

Anzahl der gestellten Themen (Aufgaben): **2**

Anzahl der Druckseiten dieser Vorlage: **20**

Bitte wenden!

Thema Nr. 1

(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Teilaufgabe 1

Aufgabe 1: Objektdiagramme (14 Punkte)

Das folgende Klassendiagramm beschreibt Modellelemente der Modellierungssprache UML und Beziehungen zwischen diesen. Kursiv geschriebene Klassennamen bezeichnen abstrakte Klassen.

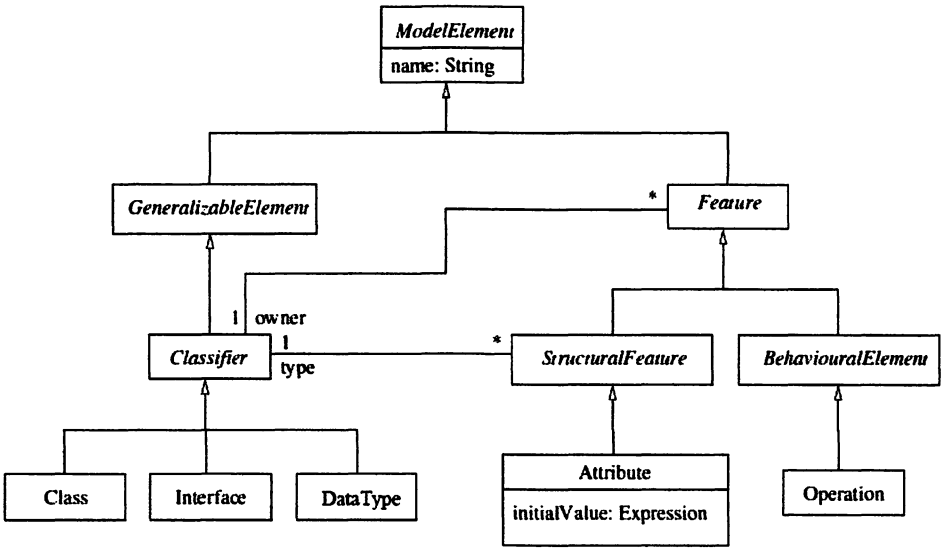
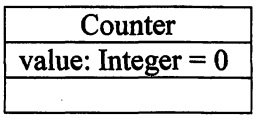


Abbildung 1: Modellelemente der UML

Ein Beispiel einer Klasse in der üblichen, grafischen UML-Notation ist die folgende Klasse **Counter**, die ein Attribut **value** vom Typ **Integer** und mit initialem Wert **0** hat.



Stellen Sie diese Klasse als UML-Objektdiagramm dar, das konform zum Klassendiagramm in Abbildung 1 gebildet ist. Für jedes Objekt ist - in der üblichen UML-Notation für Objekte - anzugeben, zu welcher Klasse es gehört, welche Werte die Attribute haben und mit welchen anderen Objekten es verbunden ist und dabei ggf. eine bestimmte Rolle spielt. Auf Objektbezeichner kann verzichtet werden.

Hinweis: **Integer** ist ein Datentyp, also als ein Objekt der Klasse **DataType** darzustellen, das den String **"Integer"** als Wert des von der abstrakten Klasse **ModelElement** geerbten Attributs **name** hat.

Fortsetzung nächste Seite!

Aufgabe 2: Klassendiagramme, STATE-Pattern, Zustandsdiagramme (12+12+19=43 Punkte)

Gegeben sei das Java-Programm in Listing 1.

a) Zeichnen Sie ein UML-Klassendiagramm, das die statische Struktur des Programms modelliert. Instanzvariablen mit einem Klassentyp sollen durch gerichtete Assoziationen mit Rollennamen und passender Multiplizität am gerichteten Assoziationsende modelliert werden. Alle aus dem Programmcode ersichtlichen statischen Informationen sollen in dem Klassendiagramm dargestellt werden.

Das Programm in Listing 1 implementiert ein Zustandsdiagramm, das das Verhalten von Objekten der Klasse `Music` beschreibt. Für die Implementierung wurde das Design-Pattern STATE angewendet.

b) Geben Sie die statische Struktur des STATE-Patterns an und erläutern Sie, welche Rollen aus dem Entwurfsmuster den Klassen des gegebenen Programms dabei zufallen und welche Operationen aus dem Entwurfsmuster durch (ggf. mehrere) Methoden in unserem Beispielprogramm implementiert werden. Es ist von den z. B. im Design-Pattern-Katalog von Gamma et al. verwendeten Namen auszugehen, das heißt von Klassen mit Namen `Context, State, ConcreteStateA, ConcreteStateB` und von Operationen mit Namen `request` und `handle`.

c) Zeichnen Sie das UML-Zustandsdiagramm (mit Anfangszustand), das von dem Programm in Listing 1 implementiert wird. Dabei muss - gemäß der UML-Notation - unterscheidbar sein, was Ereignisse und was Aktionen sind. In dem Diagramm kann zur Vereinfachung statt `System.out.println("x")` einfach `"x"` geschrieben werden.

Fortsetzung nächste Seite!

```
class Music {
  private Beatle state;
  public Music() {state = new Paul();}
  public void help() {this.state.help(this);}
  public void obladi() {this.state.obladi(this);}
  public void yesterday() {this.state.yesterday(this);}
  public void setBeatle(Beatle b) {state = b;}
}
abstract class Beatle {
  public abstract void help(Music m);
  public abstract void obladi(Music m);
  public abstract void yesterday(Music m);
}
class George extends Beatle {
  public void help(Music m) {
    System.out.println("help"); m.setBeatle(new John());}
  public void obladi(Music m) {}
  public void yesterday(Music m) {
    System.out.println("yesterday"); m.setBeatle(new Paul());}
}
class John extends Beatle {
  public void help(Music m) {
    System.out.println("help"); m.setBeatle(new Paul());}
  public void obladi(Music m) {
    System.out.println("obladi"); m.setBeatle(new Ringo());}
  public void yesterday(Music m) {}
}
class Paul extends Beatle {
  public void help(Music m) {
    System.out.println("help"); m.setBeatle(new George());}
  public void obladi(Music m) {}
  public void yesterday(Music m) {
    System.out.println("yesterday");}
}
class Ringo extends Beatle {
  public void help(Music m) {
    System.out.println("help"); m.setBeatle(new John());}
  public void obladi(Music m) {}
  public void yesterday(Music m) {
    System.out.println("yesterday"); m.setBeatle(new Paul());}
}
```

Listing 1

Fortsetzung nächste Seite!

Aufgabe 3: Systemarchitektur und Interaktionen (4+16+5=25 Punkte)

Das UML-Diagramm in Abbildung 2 zeigt eine Zwei-Schichten-Architektur eines Systems mit Benutzerschnittstelle und Anwendungskern.

a) Erläutern Sie kurz, inwiefern die gegebene Architektur die beiden wichtigen Prinzipien der hohen Kohäsion *(high cohesion)* und der losen Kopplung *(loose coupling)* erfüllt.

Im Folgenden wird ein typisches Interaktionsmuster für Objekte dieser Architektur angegeben. Zu Beginn der Kommunikation sollen existieren: Ein Benutzer als Aktor außerhalb des Systems, ein `MainWindow`-Objekt und ein `ApplicationClass`-Objekt.

1. Der Benutzer startet einen Anwendungsfall *(Use-Case)*, indem er an das `MainWindow`-Objekt die Nachricht `startUseCase` schickt.

2. Das `MainWindow`-Objekt erzeugt daraufhin ein `DialogControl`-Objekt zur Dialogkontrolle für den Use-Case.

3. Das `DialogControl`-Objekt erzeugt daraufhin ein `ApplicationControl`-Objekt sowie ein `UseCaseWindow`-Objekt (für Ein- und Ausgaben, die den Use-Case betreffen).

Damit ist das Starten des Anwendungsfalls abgeschlossen. Es folgt nun die Durchführung des Anwendungsfalls, bei der folgende Nachrichten ausgetauscht werden.

4. Der Benutzer übermittelt mit der Nachricht `enterInfoUseCase` spezifische Informationen an das `UseCaseWindow`-Objekt.

5. Die Informationen werden vom `UseCaseWindow`-Objekt mit der Nachricht `enterInfo` an das `DialogControl`-Objekt übergeben.

6. Der Übergang zum Anwendungskern erfolgt nun, indem das `DialogControl`-Objekt die Nachricht `compute` an das `ApplicationControl`-Objekt schickt.

7. Daraufhin schickt das `ApplicationControl`-Objekt die Nachricht `runApp` an das `ApplicationClass`-Objekt (das dann die eigentliche Anwendungslogik für den Use-Case durchführt).

b) Zeichnen Sie ein Sequenzdiagramm, das den beschriebenen Nachrichtenaustausch modelliert. Alle Nachrichten sollen synchron übermittelt werden. Stellen Sie in dem Sequenzdiagramm die Aktivierungsphasen der Objekte klar dar (im Unterschied zu den Zeiten, in denen ein Objekt lebt, aber nicht an einer Interaktion beteiligt ist). Die Nachrichten brauchen keine Parameter enthalten.

c) Geben Sie für jede Klasse in Abbildung 2 an, welche Operationen die Klasse anbieten muss, damit der beschriebene Nachrichtenaustausch stattfinden kann. Die Namen der Operationen sollen sich aus den Namen der Nachrichten ergeben.

Fortsetzung nächste Seite!

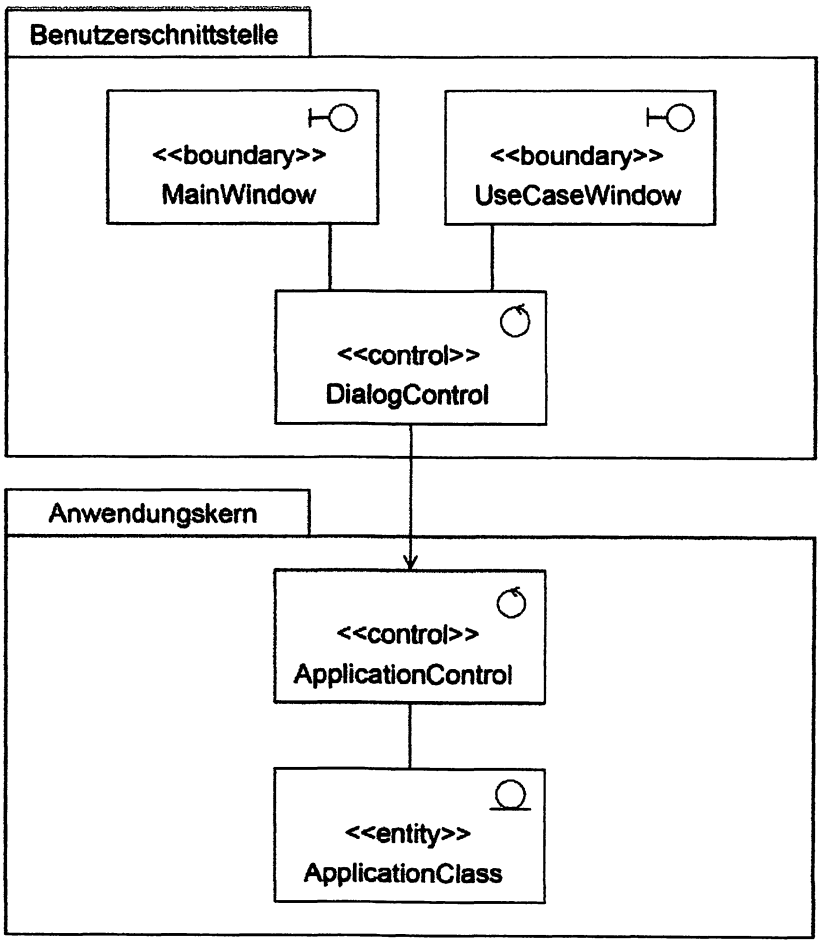
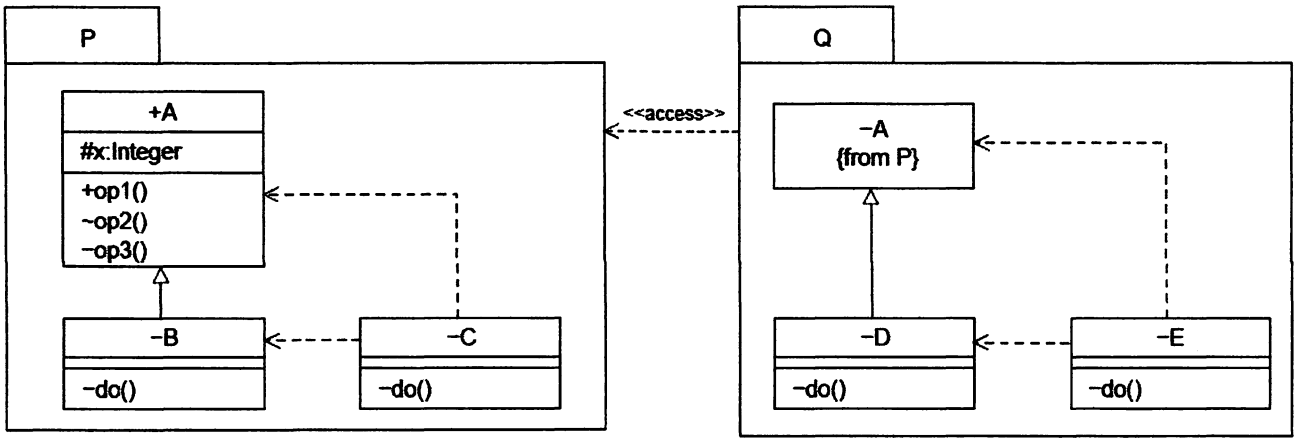


Abbildung 2: Zwei-Schichten Architektur

Fortsetzung nächste Seite!

Aufgabe 4: Sichtbarkeiten (16 Punkte)

Abbildung 3 zeigt ein statisches UML-Modell für eine Systemarchitektur mit Paketen sowie für jede der vorkommenden Klassen eine Java-Implementierung. Stellen Sie fest, welche Anweisungen in den Java-Implementierungen gemäß der angegebenen Sichtbarkeiten **unzulässig** sind und geben Sie jeweils eine kurze Begründung dafür an. Für jede falsche Antwort wird eine (vorhandene) richtige Antwort gestrichen.
Bemerkung: Für Ihre Antwort sind die im Java-Code angegebenen Sichtbarkeiten ausschlaggebend. Das UML-Modell kann jedoch hilfreich sein, um die Systemarchitektur zu überblicken. Das Symbol "~" im UML-Modell drückt Paketsichtbarkeit von Elementen einer Klasse aus. Klassen, und damit ihre nicht-privaten Elemente, sind immer im selben Paket sichtbar, auch wenn die Klasse mit "–" gekennzeichnet ist. Die <<access>>-Beziehung drückt privaten Paket-Import aus, d.h., importierte Elemente können nicht weiter exportiert werden. Die gestrichelten Linien drücken Benutzungs-Abhängigkeiten aus.




```
package P;
public class A {
  protected int x;
  public void op1() { }
  void op2() { }
  private void op3() { }
}
```

```
package P;
class B extends A {
  private void do() {
    this.op1();
    this.op2();
    this.op3();
  }
}
```

```
package P;
class C {
  private void do() {
    A a = new A();
    a.op1();
    a.op2();
    a.op3();
    B b = new B();
    b.op2();
    b.do();
  }
}
```

```
package Q;
import P.*;
class D extends A {
  private void do() {
    this.x = 1;
    this.op1();
    this.op2();
  }
}
```

```
package Q;
import P.*;
class E {
  private void do() {
    A a = new A();
    B b = new B();
    a.x = 1;
    a.op1();
    a.op2();
    A d = new D();
    d.op1();
    d.op2();
  }
}
```

Abbildung 3: Systemarchitektur mit Paketen und Java-Implementierung

Fortsetzung nächste Seite!

Aufgabe 5: Verifikation funktionaler Programme (22 Punkte)

Gegeben sei der folgende polymorphe Datentyp *'a leafTree* zur Darstellung binärer Bäume mit Blättern. Die Blätter sind mit Elementen eines beliebigen Typs *'a* markiert.

datatype *'a leafTree = Leaf* **of** *'a | Node* **of** *'a leafTree * 'a leafTree*

Die Höhe eines Baums wird durch die folgende rekursive Funktion *height* (durch Pattern-Matching gemäß dem strukturellen Aufbaus eines Baumes) berechnet. Sie berechnet die Anzahl der maximal nötigen Schritte, um von der Wurzel des Baumes zu einem Blatt zu gelangen.

fun *height(Leaf (x))* = 0
| *height(Node(T1, T2))* = *max(height(T1), height(T2))* + 1

Die Anzahl der Knoten eines Baumes (einschließlich der Blätter) wird durch die folgende rekursive Funktion *nodes* berechnet.

fun *nodes(Leaf (x))* = 1
| *nodes(Node(T1, T2))* = *nodes(T1)* + *nodes(T2)* + 1

Es wird behauptet, dass für alle Ausdrücke *T* des Typs *'a leafTree* folgendes gilt:

$$nodes(T) \leq 2^{height(T)+1} - 1.$$

Beweisen Sie diese Behauptung durch strukturelle Induktion gemäß des Aufbaus des Ausdrucks *T*. Geben Sie zunächst an, welche Fälle für die Gestalt von T unterschieden werden müssen und welcher Fall der Basisfall der Induktion ist. Für einen Nicht-Basisfall ist die Induktionsvoraussetzung zu nennen und genau anzugeben, wo diese im Beweis verwendet wird.

Fortsetzung nächste Seite!

Teilaufgabe 2

Aufgabe 1: Vermischte Fragen (2+2+2+2+2+2=12 Punkte)

Beantworten Sie die folgenden Fragen und begründen oder erläutern Sie Ihre Antwort in jeweils ein bis zwei Sätzen.

a) Was versteht man unter einem abgeleiteten Attribut?

b) Was versteht man unter referentieller Integrität?

c) Dürfen Primärschlüsselwerte NULL sein?

d) Was versteht man unter embedded SQL?

e) Was ist der Effekt des Aufrufs von commit innerhalb einer Transaktion?

f) Müssen Transaktionen von einem Datenbankverwaltungssystem, das ACID garantiert, strikt hintereinander ausgeführt werden oder dürfen mehrere Transaktionen parallel ausgeführt werden?

Aufgabe 2: ER-Modellierung (22 Punkte)

Im Folgenden finden Sie die Beschreibung eines Online-Bewerbungssystems. Erstellen Sie zu dieser Beschreibung ein erweitertes ER-Diagramm. Kennzeichnen Sie die Primärschlüssel durch passendes Unterstreichen und geben Sie die Kardinalitäten in Chen-Notation (= Funktionalitäten) an. Kennzeichnen Sie auch die totale Teilnahme (= Existenzabhängigkeit, Partizipität) von Entity-Typen.

In dem System werden Stellen ausgeschrieben. Jede Stelle hat eine eindeutige ID, eine Bezeichnung und eine Beschreibung. Stellen können notwendige Kenntnisse zugeordnet werden. Diese haben einen eindeutigen Namen und eine Beschreibung. Das System verwaltet auch Personendaten. Dabei werden Bewerber und Mitarbeiter unterschieden. Daten anderer Personengruppen sind nicht zugelassen, aber eine Person kann gleichzeitig Bewerber und Mitarbeiter sein. Alle Personen haben eine eindeutige E-Mail-Adresse, ein oder mehrere Vornamen, einen Nachnamen und ein Geburtsdatum. Bewerbern werden weiterhin Kenntnisse zugeordnet; das können auch mehrere sein. Bewerber können sich für beliebig viele Stellen bewerben. Zur Bewerbung eines Bewerbers werden ein Text und das Bewerbungsdatum gespeichert. Bewerber können sich zu einem anderen Zeitpunkt erneut auf die gleiche Stelle bewerben, aber mit einer Bewerbung immer nur auf genau eine Stelle. Einer Bewerbung kann ein zuständiger Mitarbeiter zugewiesen werden, wobei ein Mitarbeiter für beliebig viele Bewerbungen zuständig sein kann.

Fortsetzung nächste Seite!

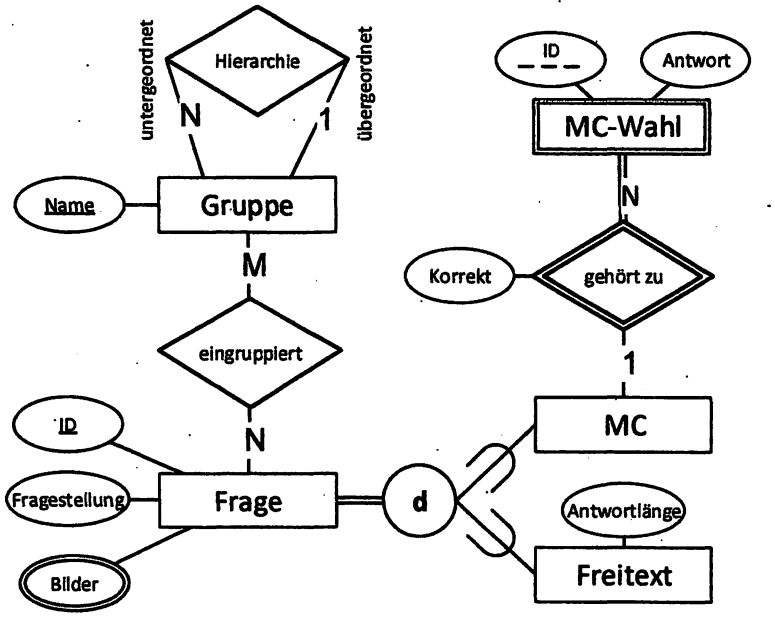
Aufgabe 3: Relationaler Entwurf (13 Punkte)

Entwerfen Sie zum untenstehenden ER-Diagramm ein Relationenschema (in dritter Normalform, 3 NF) mit möglichst wenigen Relationen.

Verwenden Sie dabei folgende Notation: Primärschlüssel werden durch Unterstreichen gekennzeichnet, Fremdschlüssel durch die Nennung der Relation, auf die sie verweisen, in eckigen Klammern hinter dem Fremdschlüsselattribut. Attribute zusammengesetzter Fremdschlüssel werden durch runde Klammern als zusammengehörig markiert. Wenn ein Attribut zur korrekten Abbildung des ER-Diagramms als `UNIQUE` oder `NOT NULL` ausgezeichnet werden muss, geben Sie dies an.

Beispiel:

```
Relation1 (Primärschlüssel, Attribut1, Attribut2,
   Fremdschlüsselattribut1[Relation2],
   (Fremdschlüssel2_Attribut1, Fremdschlüsel2_Attribut2)[Relation3]);
Attribut1 UNIQUE
```



Zwischen „Frage" einerseits und „MC" bzw. „Freitext" andererseits besteht eine Generalisierungsbeziehung. Das „d" im Kreis steht für „disjunkt", also nicht überlappende Untertypen.

Fortsetzung nächste Seite!

Aufgabe 4: SQL (5+3+3+4+6+8+7+4=40 Punkte)

Gegeben sind folgende Relationen aus einer Datenbank zur Verwaltung von Triathlon-Wettbewerben.

```
Athlet(ID, Vorname, Nachname)
Ergebnis(Athlet[Athlet], Wettbewerb[Wettbewerb], Schwimmzeit, Radzeit,
  Laufzeit); Schwimmzeit NOT NULL
Wettbewerb(Name, Jahr)
```

Verwenden Sie im Folgenden nur Standard-SQL und keine produktspezifischen Erweiterungen. Sie dürfen bei Bedarf Views anlegen. Geben Sie einen Datensatz, also eine Entity, nicht mehrfach aus.

a) Schreiben Sie eine SQL-Anweisung, die die Tabelle „Ergebnis" anlegt. Gehen Sie davon aus, dass die Tabellen „Athlet" und „Wettbewerb" bereits existieren. Verwenden Sie sinnvolle Datentypen.

b) Schreiben Sie eine SQL-Anweisung, die die Radzeit des Teilnehmers mit der ID 12 beim Wettbewerb „Zürichsee" um eins erhöht.

c) Schreiben Sie eine SQL-Anweisung, die die Namen aller Wettbewerbe des Jahres 2018 ausgibt, absteigend sortiert nach Name.

d) Schreiben Sie eine SQL-Anweisung, die die Namen aller Wettbewerbe ausgibt, in der die durchschnittliche Schwimmzeit größer als 10 ist.

e) Schreiben Sie eine SQL-Anweisung, die die IDs aller Athleten ausgibt, die im Jahr 2017 an keinem Wettbewerb teilgenommen haben.

f) Schreiben Sie eine SQL-Anweisung, die die Nachnamen aller Athleten ausgibt, die mindestens 10 Wettbewerbe gewonnen haben, das heißt im jeweiligen Wettbewerb die kürzeste Gesamtzeit erreicht haben. Die Gesamtzeit ist die Summe aus Schwimmzeit, Radzeit und Laufzeit. Falls zwei Athleten in einem Wettbewerb die gleiche Gesamtzeit erreichen, sind beide Sieger.

g) Schreiben Sie eine SQL-Anweisung, die die Top-Ten der Athleten mit der schnellsten Schwimmzeit des Wettbewerbs „Paris" ausgibt. Ausgegeben werden sollen die Platzierung (1 bis 10) und der Nachname des Athleten, aufsteigend sortiert nach Platzierung. Gehen Sie davon aus, dass keine zwei Athleten die gleiche Schwimmzeit haben und verwenden Sie **keine** produktspezifischen Anweisungen wie beispielsweise `rownum`, `top` oder `limit`.

h) Schreiben Sie einen Trigger, der beim Einfügen neuer Tupel in die Tabelle „Ergebnis" die Schwimmzeit auf den Wert 0 setzt, falls diese negativ ist.

Fortsetzung nächste Seite!

Aufgabe 5: Relationale Algebra (5+5=10 Punkte)

a) Formulieren Sie basierend auf den in Aufgabe 4 gegebenen Relationen eine Anfrage **in der Relationenalgebra**, die die IDs derjenigen Athleten ausgibt, die an allen Wettbewerben des Jahres 2015 teilgenommen haben.

b) Formulieren Sie basierend auf den in Aufgabe 4 gegebenen Relationen eine Anfrage **im relationalen Domänenkalkül**, die die Nachnamen der Athleten ausgibt, die im Jahr 2016 an mindestens einem Wettbewerb teilgenommen haben.

Aufgabe 6: Normalisierung (2+2+9=13 Punkte)

a) Erläutern Sie in etwa zwei Sätzen, wann eine Zerlegung von Relationen **verlustlos** ist.

b) Erläutern Sie in etwa zwei Sätzen, wann eine Zerlegung von Relationen **abhängigkeitsbewahrend** (abhängigkeitserhaltend) ist.

c) Erläutern Sie die drei Anomalien, die bei einem nicht-normalisierten Relationenschema auftreten können anhand von jeweils einem Beispiel. Geben Sie dazu am Besten zunächst ein nicht-normalisierte Relationenschema an und dann jeweils ein Beispiel anhand dieses Schemas.

Aufgabe 7: Optimierung (6+4=10 Punkte)

a) Übertragen Sie folgendes SQL-Statement in einen **nicht optimierten** Ausdruck der relationalen Algebra (= Anfragegraph, Operatorgraph).

```
SELECT r.id , SUM(p.anzahl)
FROM r, p
WHERE r.id = p.r
GROUP BY r.id , r.plz
HAVING SUM (p.anzahl) < 5 AND r.plz = '11037';
```

Sie können für die Aggregationsfunktion SUM in einer Attributliste SUM(r.A) mit einem qualifizierten Attribut r.A schreiben.

b) Nennen Sie zwei Möglichkeiten, den Ausdruck der relationalen Algebra aus der vorhergehenden Teilaufgabe logisch (= algebraisch) zu optimieren. Beziehen Sie sich auf konkrete Stellen und Operatoren des von Ihnen aufgestellten Ausdrucks.

Thema Nr. 2

(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Teilaufgabe 1

Aufgabe 1: Projektmanagement (10 Punkte)

Ein Team von zwei Softwareentwicklern soll ein Projekt umsetzen, das in sechs Arbeitspakete unterteilt ist. Zeichnen Sie ein Gantt-Diagramm (Balkenplan), das eine kürzestmögliche Projektabwicklung beinhaltet. Die Dauer der Arbeitspakete und ihre Abhängigkeiten können Sie aus der folgenden Tabelle entnehmen.

Name	Dauer in Wochen	Abhängig von
A1	2	-
A2	5	-
A3	2	A1
A4	5	A1, A2
A5	7	A3
A6	4	A4

Bestimmen Sie anschließend die Länge des kritischen Pfades und geben Sie an, welche Arbeitspakete an ihm beteiligt sind.

Aufgabe 2: Softwareprozesse (12 Punkte)

a) Nennen Sie vier Phasen der Softwareentwicklung in der richtigen Reihenfolge.

b) Nennen und erklären Sie kurz jeweils zwei Vorteile und zwei Nachteile beim Einsatz von Prototypen im Entwicklungsprozess.

Aufgabe 3: Softwareentwurf (19 Punkte)

Zeichnen Sie ein UML-Klassendiagramm zu folgendem Problem:
Die Verwaltung einer Universität möchte ein neues Raumverwaltungssystem entwickeln. In dem System soll für jeden Raum die Raumnummer (einfacher Integer) und die Größe in Quadratmetern gespeichert werden. Bei den Räumen soll zwischen Büros, Hörsälen und Seminarräumen unterschieden werden. Zu jedem Hörsaal und Seminarraum soll zusätzlich noch die maximale Anzahl von Sitzplätzen gespeichert werden. Außerdem soll jedes Büro mindestens einem Lehrstuhl zugeordnet sein. Es kann dabei auch vorkommen, dass ein Büro zwei oder maximal drei Lehrstühlen zugeordnet ist. Für jeden Lehrstuhl soll außerdem der Name des Lehrstuhlinhabers gespeichert werden.

Wichtig: Es müssen *keine* Methoden modelliert werden.

Fortsetzung nächste Seite!

Aufgabe 4: Design Patterns (11 Punkte)

a) Nennen Sie vier Vorteile, die die Verwendung von Design Patterns haben kann.

b) Nennen Sie drei Arten von Design Patterns und erklären Sie kurz ihren Verwendungszweck.

Aufgabe 5: Implementierung (24 Punkte)

a) Wie kann man die Lesbarkeit und Verständlichkeit des folgenden Quelltextes erhöhen? Nennen Sie vier Aspekte und geben Sie jeweils ein konkretes Beispiel anhand des gegebenen Quelltextes.

```
1  public class Calculations {
2  final static int zero=0;
3
4  /**
5   * Important computation
6   */
7  public int comp1(int BASE, int para2) {
8  if (para2<=zero) return 1;
9  return BASE * this.comp1(BASE,para2 - 1);
10 }
11 }
```

b) Lösen Sie die folgende Berechnung unter Zuhilfename einer while-Schleife (in der die Schleifenprüfung vor Ausführung des Rumpfes durchgeführt wird) und einer do-while-Schleife (in der die Schleifenprüfung nach der ersten Ausführung des Rumpfes durchgeführt wird):

Hinweise:

- Beachten Sie, dass das übergebene Array auch eine Länge von 0 haben kann.
- Sie können davon ausgehen, dass die Methode nicht mit *null* aufgerufen wird.

```
1 private static void forVariant(int[] myArray) {
2   int sum = 0;
3   for(int i = 0; i < myArray.length; i += 2) {
4     sum += myArray[i];
5   }
6   System.out.println(sum);
7 }
```

Fortsetzung nächste Seite!

Aufgabe 6: Entwicklung und Wartung (12 Punkte)

Ordnen Sie die gegebenen Wartungsaufgaben A – D jeweils einer der drei folgenden Wartungskategorien I – III zu. Begründen Sie Ihre Entscheidung kurz in einem Satz.

Wartungskategorien

(I) Corrective Maintenance (korrigierend)

(II) Adaptive Maintenance (adaptierend)

(III) Perfective Maintenance (perfektionierend)

Wartungsaufgaben

(A) Die Webseite eines Kunden soll eine Datenbankanbindung bekommen, da der Kunde sein Business-Modell geändert hat und in der Folge mit viel größeren Datenvolumina und konkurrierenden Datenzugriffen rechnet. Die Datenbankanbindung war ursprünglich nicht vorgesehen und die Daten wurden in Textdateien gespeichert.

(B) Nachdem ein Kunde auf neue Hardware und die nächste Version des Betriebssystems umgestellt hat, treten unerwartete Verhalten in den Prozessabläufen des Softwaresystems auf, die früher nicht aufgetreten sind. Die Systemumgebung wird so nicht vom Hersteller unterstützt und das System muss angepasst werden.

(C) Die Ausgabe von Prozessergebnissen läuft komplett zufriedenstellend, dennoch lässt sich die Performanz verbessern. Führen Sie die möglichen Änderungen durch.

(D) Bei der Eingabe bestimmter Zahlenfolgen stürzt das Programm ohne sichtbare Fehlermeldung ab.

Fortsetzung nächste Seite!

Aufgabe 7: Refactoring and Versionskontrolle (12 Punkte)

a) Nennen Sie sechs beliebige Refactorings, die zur Verbesserung der Lesbarkeit, Verständlichkeit und Erweiterbarkeit eines Softwaresystems durchgeführt werden können.

b) Diskutieren Sie anhand der folgenden Projekt-Beschreibung was für den Einsatz eines zentralen oder dezentralen Versionskontrollsystems spricht. Begründen Sie Ihre Entscheidung.

Wichtig: Es geht bei dieser Entscheidung nur um *zentral* oder *dezentral*, es sollen keine konkreten Versionskontrollsysteme (wie bspw. GIT oder SVN) betrachtet werden.

Eine Softwarefirma möchte ein neues Softwareentwicklungsprojekt umsetzen und muss sich entscheiden, ob es ein zentrales oder dezentrales Versionskontrollsystem einsetzt. Es gibt dabei folgende Dinge im Entwicklungsprozess zu beachten:

- Die Entwickler dürfen Home-Office machen.
- Einige Entwickler sind viel auf Reisen und deswegen nicht immer mit dem Intranet verbunden.
- Es gibt einen zentralen Build-Server im Intranet der Firma, der sich aus einer Kopie des Repositories speist. Es ist wegen der technischen Details nicht möglich, Tests auf den Arbeitsplatzrechnern der Entwickler laufen zu lassen.
- Features werden auf Branches entwickelt, der Build-Server testet stets nur einen bestimmten Branch, den die Entwickler in Absprache untereinander konfigurieren können.

Aufgabe 8: Verifikation, Validierung und Testing (8 Punkte)

a) Erklären Sie in wenigen Sätzen den Unterschied zwischen Fehlervermeidung und Fehlertoleranz.

b) Was versteht man unter Regression-Testing und welche Anforderungen sind an Regressionstests gestellt?

Aufgabe 9: Softwarequalität (4 Punkte)

Warum sollte Qualitätsmanagement vom Projektmanagement getrennt werden?

Aufgabe 10: Softwaremaße (8 Punkte)

Nennen Sie drei Kategorien von einfachen Softwaremaßen und beschreiben Sie kurz, wie mithilfe dieser die Komplexität von Softwareprojekten ermittelt wird.

Fortsetzung nächste Seite!

Teilaufgabe 2

Aufgabe 1: Datenmodellierung mit ER-Modellen (15+4+12+3=34 Punkte)

Wir betrachten eine Datenbank, die einem Sportverband (z.B. dem *Deutschen Fußballbund*) zur Verwaltung seiner Unterverbände, Vereine, sowie deren Mitglieder und Funktionsträger dient. Sie verfügt über die folgenden Eigenschaften:

- Für jeden (Unter-)Verband werden sein Name sowie eine eindeutige Verbandsnummer abgespeichert. So gibt es beispielsweise den *Bayerischen Fußballverband.*
- Die Verbände sind hierarchisch organisiert, d.h. zu jedem Verband kann sein Dachverband angegeben werden. Für die höchste Ebene (im Beispiel *Deutscher Fußballbund*) wird kein Dachverband hinterlegt.
- Jeder Verein gehört genau einem Verband an. Ein Verein besitzt eine Vereinsnummer, die zusammen mit der Verbandsnummer eindeutig ist. Zum Verein werden der Vereinsname (z.B. *FC Bayern München*) sowie das Gründungsdatum abgespeichert.
- Eine Person ist gekennzeichnet durch eine eindeutige Nummer sowie einen Vor- und einen Nachnamen.
- Eine Person kann Mitglied in beliebig vielen Vereinen sein. Zu jeder Mitgliedschaft sollen Beginn und Ende abgespeichert werden.
- Es gibt verschiedene Funktionen, wie z.B. *Vorsitzende/r* oder *Kassierer/in.* Ihre Bezeichnung ist eindeutig. Zu jeder Funktion wird abgespeichert, ob sie nach Vereinsrecht notwendig ist oder nicht. Für die Verbände und Vereine können Personen mit ihrer Verbands- bzw. Vereinsfunktion hinterlegt werden. So hat bspw. *Uli Hoeneß* beim *FC Bayern München* die Vereinsfunktion als Präsident inne.

Mögliche Dateneinträge sind *kursiv* hervorgehoben und dienen einzig der Veranschaulichung.

a) Modellieren Sie das oben genannte Datenbanksystem möglichst vollständig in einem erweiterten ER-Diagramm. Kennzeichnen Sie die Schlüssel der Entity-Typen und geben Sie für die Relationship-Typen jeweils Funktionalitätsbedingungen (in N:M-Notation) oder Kardinalitätsbedingungen (in (MIN, MAX)-Notation) an.

b) Was ist eine schwache Entity? Ist der Schlüssel einer schwachen Entity global eindeutig? Falls nein, wie entsteht ein global eindeutiger Schlüssel für eine solche Entity?

c) Übertragen Sie Ihr ER-Modell in das relationale Datenmodell. Erstellen Sie dazu die Tabellen `Verband`, `Verein` und `Person` mit Hilfe von CREATE TABLE Statements in SQL. Berücksichtigen Sie die Fremdschlüsselbeziehungen, und geben Sie für diese explizit die Update- und Delete-Modi über `ON UPDATE` bzw. `ON DELETE` an. Wählen Sie sinnvolle Datentypen.

Fortsetzung nächste Seite!

d) Was ist zu beachten, wenn in SQL zu einem Verein all seine Dachverbände bestimmt werden sollen? Sie müssen die SQL-Anfragen nicht angeben, Sie sollen nur die auftretende Problematik beleuchten.

Aufgabe 2: SQL (6+3+3+2,5+4,5+3+5+2+8 = 37 Punkte)

Wir betrachten eine Datenbank zur Speicherung der Fahrplandaten von Fernverkehrszügen der Deutschen Bahn. Die Datenbank enthält die Tabellen BAHNHOF, ZUG und HALT, deren Inhalt im Folgenden in Auszügen angegeben ist:

BAHNHOF		
Bhf	Name	Gleise
AH	Hamburg Hbf	8
BH	Berlin Hbf	14
FF	Frankfurt Hbf	24
HH	Hannover Hbf	12
MH	München Hbf	32
NN	Nürnberg Hbf	22
TS	Stuttgart Hbf	16

HALT			
Zug_Nr	Bhf	An	Ab
ICE 882	MH	-	08:22
ICE 882	NN	09:30	09:33
ICE 882	HH	12:32	12:36
ICE 882	AH	13:54	-
ICE 595	BH	-	07:12
ICE 595	FF	11:44	11:50
ICE 595	TS	13:46	13:50
ICE 595	MH	16:04	-
ICE 778	FF	-	16:34

ZUG	
Zug_Nr	Wagen
ICE 882	12
ICE 595	10
ICE 778	8

a) Entwerfen Sie passend zu diesen Tabellen ein erweitertes ER-Diagramm. Geben Sie für die Relationship-Typen jeweils sinnvolle Funktionalitätsbedingungen (in N:M-Notation) oder Kardinalitätsbedingungen (in (MIN, MAX)-Notation) an.

b) Geben Sie das Resultat der folgenden SQL-Anweisung an:

```
SELECT Zug.Zug_Nr, SUM(Wagen) AS Summe FROM Halt, Zug
WHERE Halt.Zug_Nr = Zug.Zug_Nr
GROUP BY Zug.Zug_Nr;
```

c) Geben Sie das Resultat der folgenden SQL-Anweisung an:

```
SELECT Bhf, COUNT(*) AS Gleise FROM Halt
GROUP BY Bhf
HAVING Gleise != 1;
```

d) Schreiben Sie eine SQL-Anweisung, die den HALT für *ICE 595* in *Hannover Hbf* einfügt (Ankunft 09:11 Uhr, Abfahrt 09:13 Uhr).

e) Schreiben Sie eine SQL-Anweisung, die einen View erzeugt, der zu jeder Zugnummer die Anzahl der angefahrenen Halte enthält.

Fortsetzung nächste Seite!

f) Schreiben Sie eine SQL-Anweisung, die alle Züge löscht, für die keine Halte gespeichert sind.

g) Schreiben Sie eine SQL-Anweisung, die die Namen aller Bahnhöfe zurückliefert, deren Gleisanzahl kleiner ist als die durchschnittliche Gleisanzahl aller Bahnhöfe. Die Bahnhöfe sollen aufsteigend nach ihrer Gleisanzahl ausgegeben werden, bei Gleichheit alphabetisch sortiert.

h) Schreiben Sie eine SQL-Anweisung, die alle Wagenanzahlen der Züge um eins erhöht.

i) Schreiben Sie eine SQL-Anweisung, die die Zugnummern all jener Züge zurückliefert, die im Laufe einer Fahrt den selben Bahnhof mehrfach anfahren. Geben Sie jedes Paar von Zugnummer und Bahnhofskürzel nur einmal aus.

Aufgabe 3: Funktionale Abhängigkeiten und Zerlegungen (5+5+5+9+4+6=34 Punkte)

Gegeben sei das Relationenschema $R = (U; F)$ mit der Attributmenge U und der Menge F von funktionalen Abhängigkeiten (fds):

$$U = \{A, B, C, D, E, G, H, I\};$$
$$F = \{\{B, C\} \rightarrow \{E\}, \{B, C, G\} \rightarrow \{H\}, \{D\} \rightarrow \{A, I\}, \{E, G\} \rightarrow \{H\}\}.$$

a) Zeigen Sie, dass die fd $\{B, C, G\} \rightarrow \{H\}$ in F redundant ist, d.h., dass sie bereits aus den drei anderen fds von F folgt.

b) Geben Sie alle Schlüssel für das Relationenschema R sowie die Nichtschlüsselattribute an (mit Begründung).

c) Ist R in 3NF, ist R in BCNF (mit Begründung)?

d) Geben Sie eine Basis (kanonische Überdeckung) G von F an und führen Sie damit die 3NF-Synthese aus.

e) Welches Ergebnis würde der Algorithmus 3NF-Synthese liefern, der fds in der Basis mit gleichen linken Seiten zusammenfasst?

f) Geben Sie eine erlaubte Instanz r zu R mit mindestens 4 Tupeln an, so dass die funktionale Abhängigkeit $\{B, C\} \rightarrow \{H\}$ nicht gilt (mit Begründung).

Hinweis: Wenn Sie die Attributwerte für B bzw. E alle konstant auf b bzw. e setzen und die Werte der Attribute $X \in \{A, D, G, H, I\}$ tupelweise durchnummerieren (d.h. X hat den Wert x_i in Tupel i), dann müssen Sie nur geeignete Werte für das Attribut C finden.

Fortsetzung nächste Seite!

Aufgabe 4: Transaktionen (8+7=15 Punkte)

Gegeben seien die folgenden beiden Schedules 1 und 2.

- Schedule 1:

T_1	T_2	T_3	T_4	T_5
				$read(A)$
$read(B)$				
$read(A)$				
				$write(A)$
			$read(C)$	
			$write(C)$	
		$read(A)$		
$write(B)$				
	$read(B)$			
	$write(B)$			
		$read(C)$		

- Schedule 2:

T_1	T_2	T_3
	$read(X)$	
$read(Z)$		
$write(Z)$		
		$read(Y)$
	$read(Y)$	
	$write(X)$	
	$write(Y)$	
		$write(Y)$

In beiden Schedules sollen alle Transaktionen am Ende erfolgreich beendet werden (commit).

a) Prüfen Sie Schedule 1 auf konfliktbasierte Serialisierbarkeit. Geben Sie eine Begründung und eine eventuelle Serialisierungsreihenfolge an.

b) Prüfen Sie Schedule 2 auf konfliktbasierte Serialisierbarkeit. Geben Sie eine Begründung und alle möglichen Serialisierungsreihenfolgen an.