TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Software- und Multimediatechnik, Professur Softwaretechnologie

Technische Universität Dresden, 01062 Dresden

Prof. Dr.rer.nat.habil.
Uwe Aßmann

Klausur
Softwaretechnologie SS 2017

Name:	
Vorname:	
Immatrikulationsnummer:	

Aufgabe	Maximale Punktanzahl	Erreichte Punktanzahl
1	**25**	
2	**4**	
3-I	**4**	
3-II	**19**	
3-III	**38**	
Gesamt	**90**	

Hinweise:

- In der Klausur ist als Hilfsmittel lediglich ein **A4-Blatt, beidseitig beschrieben,** zugelassen.
- Die Klammerung der Aufgabenblätter darf **nicht** entfernt werden.
- Tragen Sie bitte die Lösungen auf den Aufgabenblättern ein!
- Verwenden Sie keine roten, grünen Stifte oder Bleistifte!
- Es ist kein eigenes Papier zu verwenden! Bei Bedarf ist zusätzliches Papier bei der Aufsicht erhältlich. Bitte jedes zusätzliche Blatt mit Name, Vorname und Immatrikulationsnummer beschriften.
- Es sind alle Aufgabenblätter abzugeben!
- Ergänzen Sie das Deckblatt mit Name, Vorname und Immatrikulationsnummer!
- Halten Sie Ihren Studentenausweis und einen Lichtbildausweis zur Identitätsprüfung bereit.

Aufgabe 1: Modulchaos (25 Punkte)

Um die vielen Module in den verschiedenen Studiengängen an einer Universität zu verwalten, soll ein Informationssystem aufgebaut werden. Dazu müssen zunächst die dafür relevanten Objekte und deren Beziehungen in einem Domänenmodell erfasst werden. Es existiert folgende textuelle Beschreibung:

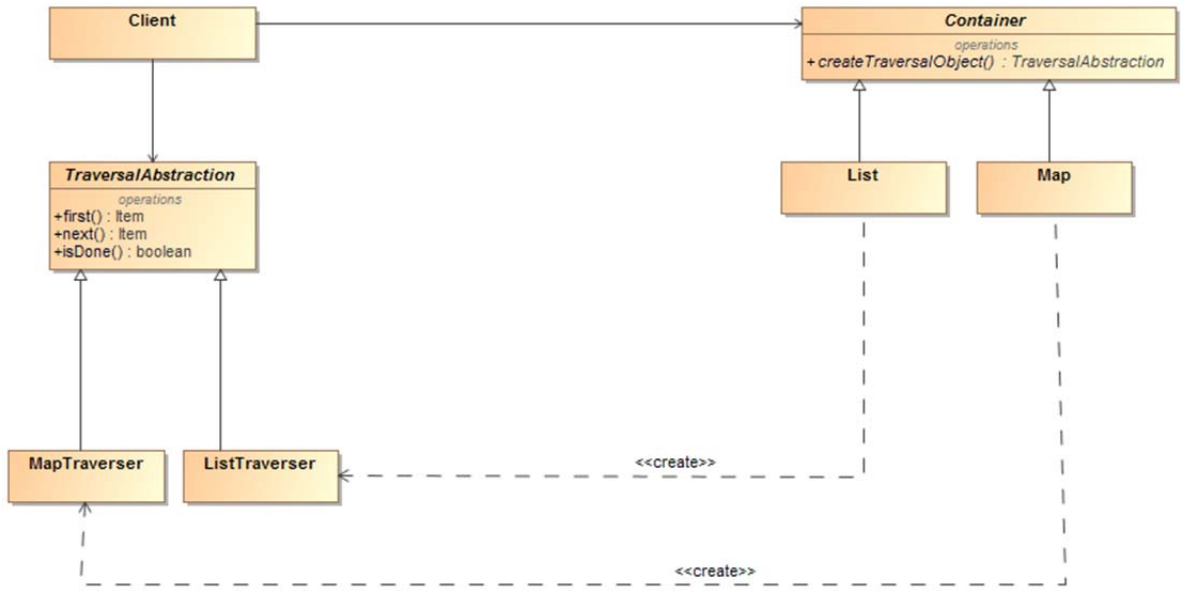
Ein *Modul* ist durch einen *Identifikator* und eine ausführliche *Beschreibung* gekennzeichnet. Jedes *Modul* kann Teil verschiedener *Studienordnung*en sein.
Eine *Studienordnung* regelt genau einen *Studiengang*, wobei es für einen *Studiengang* mehrere *Studienordnung*en geben kann, die jeweils aus einem anderen *Jahr* stammen. Jede *Studienordnung* enthält mindestens drei *Modul*e.
Jeder *Studiengang* wird an einer *Fakultät* verwaltet. Die *Studiengäng*e werden nach ihrem *Abschluss* (*Bachelor*, *Master*, *Diplom*, *Lehramt*) unterschieden.
*Lehrveranstaltung*en müssen mindestens einem *Modul* zugeordnet werden. Jedes *Modul* kann mehrere, muss jedoch mindestens eine *Lehrveranstaltung* referenzieren. Zusätzlich müssen die *Anzahl SWS* (SWS) und die *Anzahl Leistungspunkte* (LP) erfasst werden. Es ist dabei zu beachten, dass es oft vom *Modul* abhängt, wie viele SWS und LP einer *Lehrveranstaltung* in einen *Modul* eingebracht werden.
Das *Lehrangebot* eines *Semester*s regelt, welche der *Lehrveranstaltung*en in einem *Semester* angeboten werden.
Zusätzlich werden die agierenden *Person*en erfasst. Eine solche *Person* kann sowohl *Modulverantwortlicher* als auch *Lehrbeauftragter* für eine oder auch mehrere *Lehrveranstaltung*en sein. Für jedes *Modul* ist genau ein *Modulverantwortlicher* bekannt, eine *Person* kann aber *Modulverantwortlicher* für mehrere *Modul*e sein. Für jede *Lehrveranstaltung* muss es mindestens einen *Lehrbeauftragten* geben.
Jede *Person* ist einer *Professur* zugeordnet. Außerdem wird der Aufbau der *Fakultät*en abgebildet. Jede *Fakultät* besteht aus mindestens zwei *Institut*en und jedes *Institut* aus mindestens zwei *Professur*en. Falls *Fakultät*en aufgelöst werden, werden auch die dazugehörigen *Institut*e aufgelöst. In diesem Fall wird eine dazugehörige *Professur* eines *Institut*es einer anderen Struktureinheit der Universität zugeordnet.

Erstellen Sie das Domänenmodell (UML-Analyse-Klassendiagramm)!
Berücksichtigen Sie dabei folgende Hinweise:

- **Denken Sie an Klassen, Enumerationen, Klassenbeziehungen, Assoziationsklassen, Multiplizitäten, Rollennamen und Attribute!**
- **Es sollen keine Operationen modelliert werden!**
- **Alle domänenspezifischen Begriffe, die im obigen Text *kursiv* geschrieben sind, müssen im Modell wiederzufinden sein!**

Aufgabe 2: Entwurfsmuster (4 Punkte)

Sie müssen nacheinander alle Elemente einer aggregierten Struktur (z.B. Map oder List) bearbeiten (siehe unten stehendes Entwurfsklassendiagramm). Dazu müssen diese sequenziell geliefert werden, wobei unterschiedliche Traversierungsvarianten zum Einsatz kommen. Die entsprechende Logik sollte außerhalb des Clients realisiert werden, um diesen von der Struktur zu entkoppeln.
Um welches Muster handelt es sich? Zeichnen Sie das Entwurfsmuster mit den beteiligten Rollen als UML-Kollaboration in das Diagramm ein!



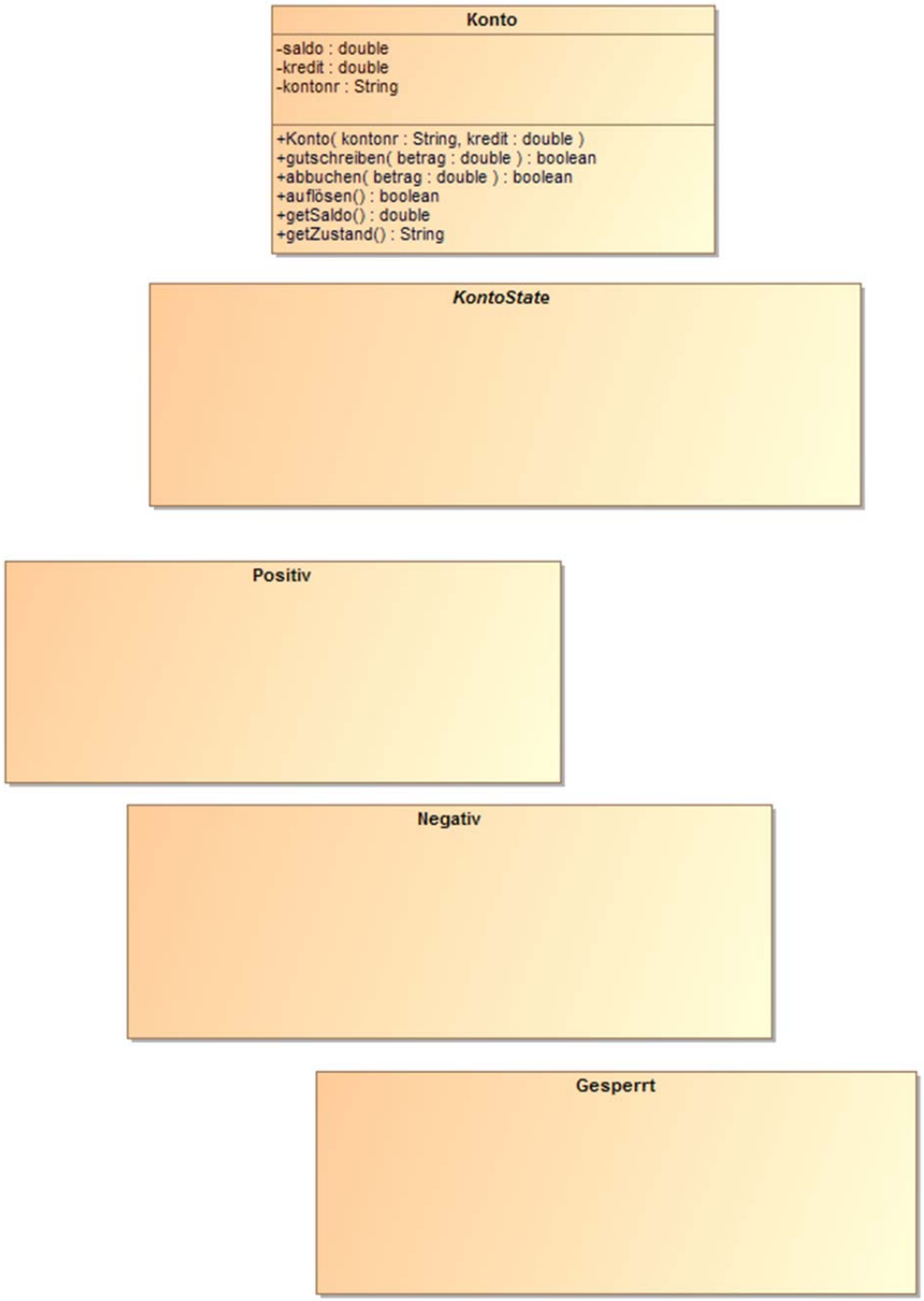
Aufgabe 3: Konto (61 Punkte)

Im Folgenden ist ein Konto beschrieben (Tipp: Erinnern Sie sich an die entsprechende Übungsaufgabe!):
Ein Konto wird eingerichtet (*einrichten*) und hat einen anfänglichen Kontostand (*Saldo*) gleich Null. Ist der Kontostand eines Kontos größer oder gleich Null, so ist das Konto „positiv“. Ist der Kontostand kleiner als Null, so ist das Konto „negativ“. Der Kontostand wird durch den Vorgang *abbuchen* verringert und durch den Vorgang *gutschreiben* erhöht. Ein Konto kann maximal um die Höhe des festgelegten Überziehungskredites (*Kredit*) überzogen werden (unterstes Limit). Wenn der Kontostand das unterste Limit erreicht hat, ist das Konto bis zum nächsten Vorgang *gutschreiben* gesperrt. Ein Konto kann nur mit einem Kontostand gleich Null aufgelöst (*aufloesen*) werden.

Zusätzlich ist ein UML-Modell (Entwurfsklassendiagramm und Protokollzustandsmachine) gegeben. Die Protokollmaschine ist auf einem Extrablatt zu finden.

- Das Ereignis *einrichten* wird im Entwurf mit den `Konto-`Konstruktor modelliert.
- Die Methoden `gutschreiben()`und `abbuchen()`geben genau dann `false` zurück, falls sich der Saldo nicht verändert, d.h. die Aktion nicht ausgeführt werden konnte. Anderenfalls wird `true` zurückgegeben.
- Die Methode `aufloesen()` gibt genau dann `true` zurück, wenn das Konto aufgelöst werden kann, ansonsten `false`.

- Der Entwurf verwendet das State Pattern.



Lösen Sie folgende Teilaufgaben:

I. Ergänzen Sie das Entwurfsklassendiagramm um die Beziehungen zwischen den Klassen und die fehlenden Methoden in den Klassen `KontoState` (abstrakte Klasse), `Positiv`, `Negativ` und `Gesperrt`! Tragen Sie alle Methoden mit Parametern und Rückgabewert ein. Methoden, die nicht überschrieben werden, müssen weggelassen werden. (4 Punkte)

II. Überlegen Sie sich jeweils einen „Gut“(ok)-Testfall mit

`saldo = -100.00` und `kredit = 500.00`

für die Methoden in der unten stehenden Testtabelle!

Diese Testfälle sollen

a) jede ausgehende Transition des Zustandes `negativ` (Äquivalenzklassenbildung, Transition 1 bis 6) und
b) den Grenzfall, dass der sich ergebende Saldo gleich Null ist (Transition 7)

überdecken.

Tragen Sie diese Testfälle mit den entsprechenden Werten in die Testfalltabelle ein!

Ausgehende Transitionen im Zustand `Negativ`			Ausgabe (expected) der Testfälle		
Transition #T	Call (Ereignis bzw. Methodenaufruf im Fall `saldo = -100` `kredit = 500.00`)	`betrag`	Rückgabe-wert (`boolean`) des Calls #M1	`getSaldo()` (`double`) #M2	`getZustand()` (`String`) #M3
1					
2					
3					
4					
5					
6					
7				`0`	

c) Ergänzen Sie für einen `gutschreiben()`/#M2- Testfall (`getSaldo()`) die folgende jUnit Testmethode:

```
private Konto konto;
@Test
public void test_Tx_M2 () {




}
```

(19 Punkte)

III. Implementieren Sie die Klasse `Konto` entsprechend dem UML-Modell unter Verwendung des State-Entwurfsmusters!

- **Nutzen Sie dazu innere Klassen!**
- **Sie können davon ausgehen, dass der Parameter `betrag` immer größer als Null ist.**

(38 Punkte)

```
public class Konto {
```

```
public String getZustand() {
    //gibt den Namen des Zustands des Kontos zurück
    //muss nicht implementiert werden
}


class Gesperrt …
    // muss nicht implementiert werden
}
```