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 WS 2014/15

Name:	
Vorname:	
Immatrikulationsnummer:	

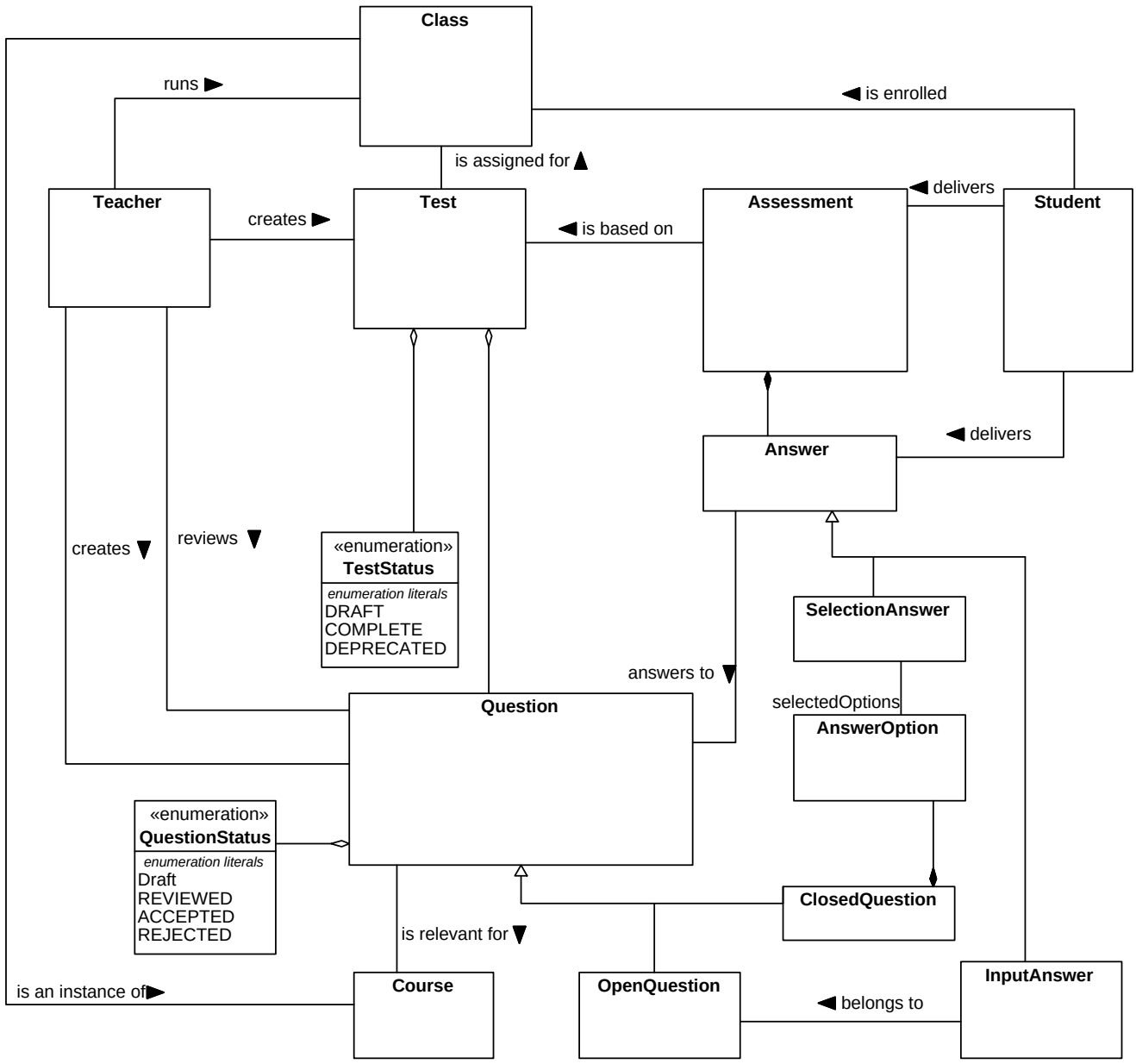
Aufgabe	Maximale Punktanzahl	Erreichte Punktanzahl
1	**33**	
2	**7**	
3	**12**	
4	**38**	
Gesamt		

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.
- **Achtung!** Das Zeichen ✍ heißt: **Hier ist Java-Text einzufügen**!

Aufgabe 1: E-Assessment (33 Punkte)

Eine Fakultät möchte elektronische Klausuren einführen. Dazu wird ein Konzept ausgearbeitet. Begonnen wird mit der Modellierung der Domäne. In der ersten Runde entsteht das folgende Analyseklassendiagramm.



Es werden (elektronische) Klausuren (*Test*) und Prüfungen (*Assessment*) modelliert. Unter einer Prüfung wird die Prüfung eines Studenten (*Student*) einschließlich der Bewertung der Prüfung verstanden, die der Student in einer (*is based on*) Klausur geliefert (*delivers*) hat.

Ein Student nimmt in der Regel an mehreren Klausuren teil. Ein Student kann mehrere Lehrveranstaltungen (*Course*) besuchen. Zu jeder Lehrveranstaltung gibt es mindestens eine Seminargruppe (*Class*), in die sich ein Student einschreiben kann (*is enrolled*). Eine Seminargruppe wird von mindestens zwei und maximal 30 Studenten besucht. In der Regel betreut (*runs*) ein Lehrbeauftragter (*Teacher*) mehrere Seminargruppen. Jede Seminargruppe wird von genau einem Lehrbeauftragten betreut.

Der Prozess des Entwurfs einer Klausur ist aufwendig. Es werden zunächst Fragen (*Question)* als *DRAFT* entwickelt. Jede Frage ist für mindestens eine Lehrveranstaltung relevant (*is relevant for*) und wird von genau einem Lehrbeauftragten geschrieben (*creates*). Typischerweise sind verschiedene Lehrbeauftragte daran beteiligt und jeder dieser Lehrbeauftragten kann sich mehrere Fragen überlegen. In einem nächsten Schritt wird jede Frage von oft mehreren Kollegen (*Teacher*) begutachtet (*reviews*). Eine begutachtete (*REVIEWED*) Frage wird nach einer Diskussion unter den Lehrbeauftragten akzeptiert (*ACCEPTED*) oder abgelehnt (*REJECTED*).

Da jede Klausur aus Rechnerkapazitätsgründen für jede Seminargruppe neu organisiert werden muss, stellt (*creates*) ein Lehrbeauftragter eine Klausur, bestehend aus mindestens einer Frage, zusammen und weist diese Klausur genau einer Seminargruppe zu (*is assigned for*).

Es werden offene (*OpenQuestion*) und geschlossene (*ClosedQuestion*) Fragen unterschieden. Die Beantwortung einer offenen Frage erfordert die Eingabe eines Textes. Eine geschlossene Frage dagegen ist eine Multiple-Choice-Aufgabe, für die in der Aufgabenstellung insgesamt mindestens fünf Antwortoptionen (*AnswerOption*) vorgegeben werden.

Die Prüfung eines Studenten (*Assessment*, siehe oben) besteht aus den Antworten (*Answer*) des Studenten, die er auf die Fragen der Klausur liefert (*delivers*). Eine Antwort wird auch als eine Antwort gewertet, wenn diese vom Studenten nicht bearbeitet wurde. Jeder Student kann nur eine Antwort (*Answer*) auf eine Frage geben. Die Antwort, die zu einer offene Frage gehört (*belongs to*), heißt *InputAnswer*. Im Fall der Antwort (*SelectionAnswer*) auf eine geschlossene Frage wählt der Student die aus seiner Sicht richtigen Antwortoptionen (*selectedOptions*) aus.

Teilaufgabe 1.1:

Ergänzen Sie entsprechend der oben gegebenen Beschreibung im Analyseklassendiagramm alle Klassenbeziehungen um die richtigen Multiplizitäten!

- **Falls eine Multiplizität *direkt* oder *indirekt* nicht aus dem Text ableitbar ist, kann sie weggelassen werden.**

Teilaufgabe 1.2:

Ergänzen Sie das Analyseklassendiagramm auf Seite 2 um weitere domänenspezifische Informationen:

Einer Seminargruppe einer Lehrveranstaltung werden genau eine Zeit und ein Raum zugeordnet. Für jede Klausur müssen ein Titel, eine erlaubte Bearbeitungszeit sowie eine Instruktion zur Durchführung der Klausur angegeben werden.

Jede Frage braucht eine Beschreibung sowie die maximale Anzahl von erreichbaren Punkten. Für eine offene Frage wird die erwartete Antwort angegeben. Im Fall einer geschlossenen Frage wird für jede ihrer Antwortoptionen ein Antworttext, und es wird zusätzlich angegeben, ob der Antworttext richtig ist.

Die Note der Prüfung (*Assessment*) eines Studenten berechnet sich aus den Punkten, die er durch seine Antworten erzielen konnte. Jeder Student wird durch seine Matrikelnummer identifiziert. Zusätzlich werden sowohl der Student als auch der Lehrbeauftragte durch Namen und Emailadresse erfasst.

Teilaufgabe 1.3:

In einer zweiten Runde der Begutachtung des obigen Analyse-Klassendiagramms wird festgestellt, dass die Semantik einiger Klassen besser durch Assoziationsklassen modelliert werden kann. Modellieren Sie für ein Beispiel den entsprechenden Ausschnitt aus dem Analyseklassendiagramm neu:

Teilaufgabe 1.4:

Die Lehrbeauftragte Birgit hat die offene Frage OQ1 für die Lehrveranstaltung SWT geschrieben. Eine weitere Frage (die geschlossene Frage CQ1) wurde vom Lehrbeauftragten Max geschrieben. Der Lehrbeauftragte Georg hat beide Fragen begutachtet. Die geschlossene Frage CQ1 wurde zusätzlich von Birgit begutachtet. Bislang wurde aber nur die Frage OQ1 für die Verwendung in einer Klausur akzeptiert. Birgit erstellt eine Klausur K2015 und weist dieser als erste Frage OQ1 zu.

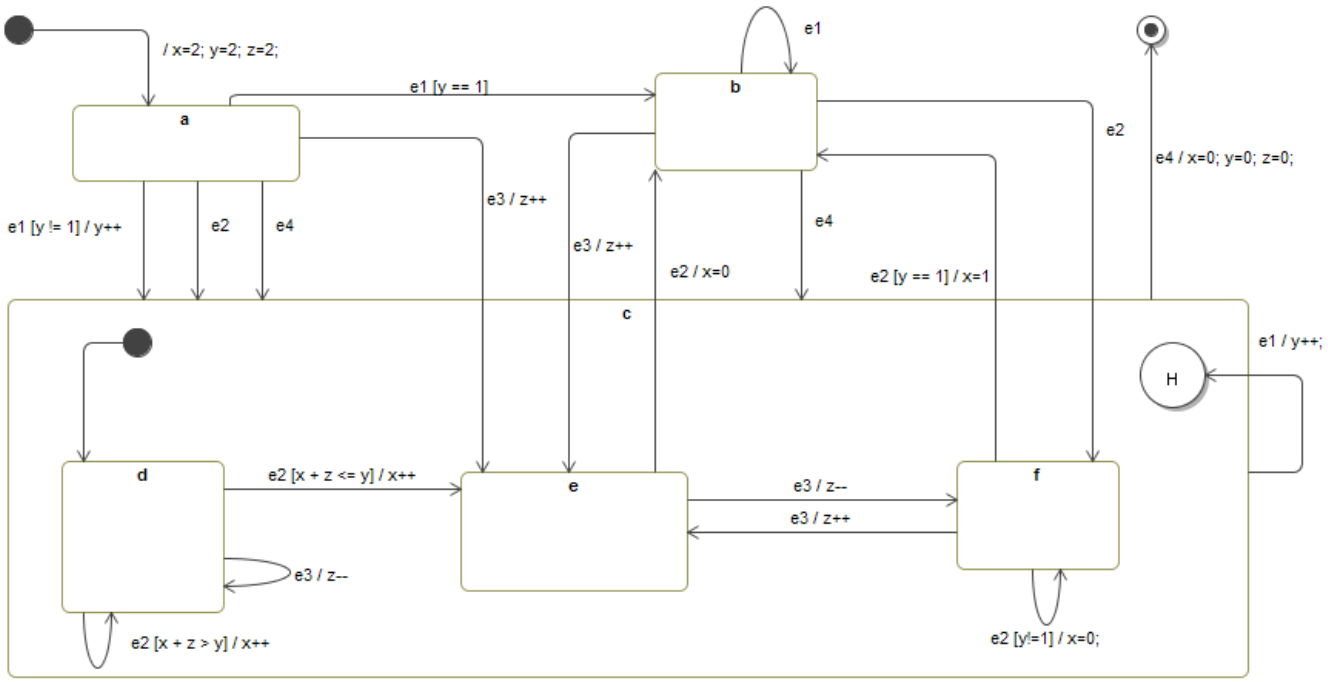
Erstellen Sie ein Objektdiagramm, welches oben beschriebenes Objektnetz beschreibt!

- **Beachten Sie, jedes Objekt durch seinen Objekt- und Klassennamen und jeden Link mit seinem zugehörigen Assoziationsnamen zu beschriften (sofern dieser im Ausgangsmodell vorhanden ist).**

Aufgabe 2: Abstraktes Zustandsmodell (7 Punkte)

Gegeben ist das folgende (abstrakte) Zustandsmodell. Bedingungen (Guards) und Aktionen sind in Java-Syntax spezifiziert.

Füllen Sie die folgende Tabelle aus, in dem Sie das Zustandsmodell in der Reihenfolge der gegebenen Ereignisse abarbeiten und jeweils den neuen Zustand und die Belegung der Variablen angeben!



Ereignis-Nr.	Ereignis	Neuer Zustand	Variablenwerte		
			x	y	z
1	start	a	2	2	2
2	e1				
3	e1				
4	e3				
5	e2				
6	e2				
7	e4				
8	e4				

<u>Aufgabe 3:</u> UML-Diagramme (12 Punkte)

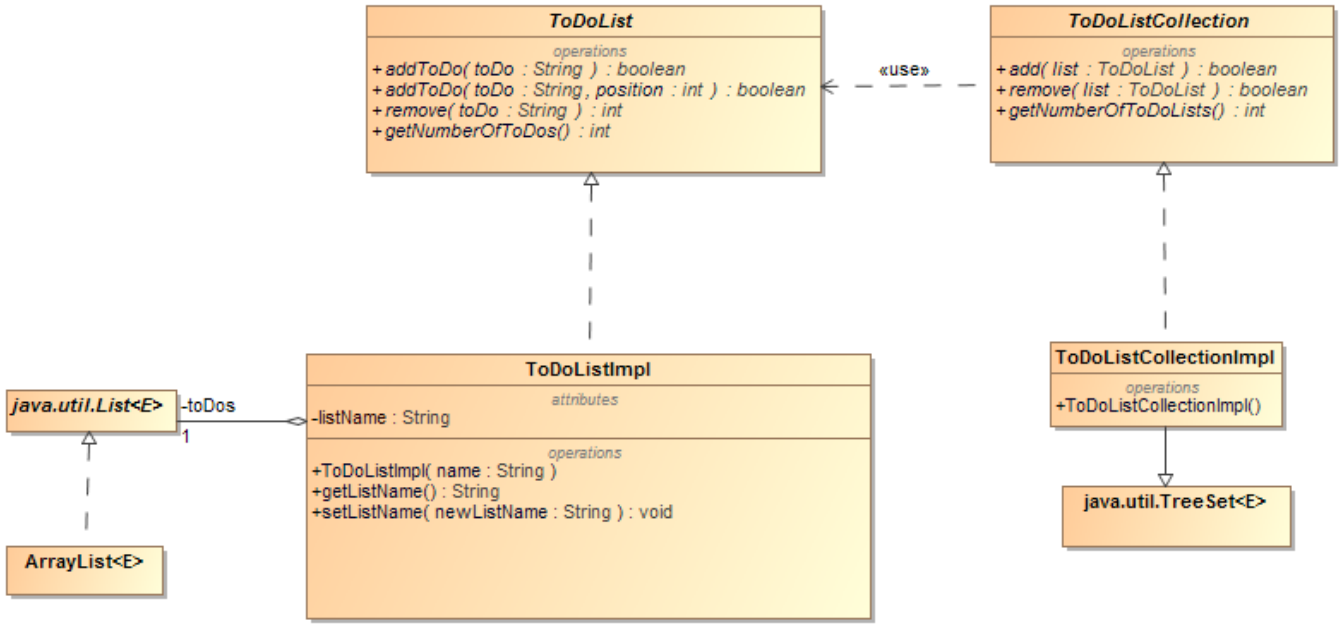
Die Qualität und Nützlichkeit von Modellen und UML-Diagrammen hängt entscheidend von deren Einsatz in den Phasen im Softwarelebenszyklus sowie der Charakteristik der Projekte ab.

Beschreiben Sie in der folgenden Tabelle, wofür die Modellierung mit den folgenden UML-Diagrammarten geeignet ist und kreuzen Sie an, in welchen Phasen des Software-Lebenszyklus (Analyse (OOA) und/oder Entwurf (OOD)) die Diagramme zum Einsatz kommen!

UML-Diagramm	Beschreibung	Phase	
		OOA	OOD
Anwendungsfall-diagramm			
Analyseklassen-diagramm			
Entwurfsklassen-diagramm			
Objektdiagramm			
Verhaltenszustands-maschine			
Protokollmaschine			
Aktivitäts-diagramm			
Sequenz-diagramm			

Aufgabe 4: ToDo-Listen (38 Punkte)

Eine ToDo-Liste beinhaltet in geordneter Reihenfolge „toDos“. Die ToDo-Listen einer Person werden in einer Kollektion verwaltet. Der Entwurf für die Implementierung von ToDo-Listen sieht vor, dass es ein *Interface* für eine ToDo-Liste (`ToDoList`) und ein *Interface* für eine Kollektion von ToDo-Listen (`ToDoListCollection`) gibt. Die Interfaces und deren Implementationen sind im folgenden Entwurfsklassendiagramm beschrieben.



Eine **`ToDoListImpl`** hat eine Bezeichnung (`listName`) und besteht aus den eigentlichen `toDos` (angeordnet in einer Liste: `List<E>`). Eine Kollektion von ToDoListen (**`ToDoListCollectionImpl`**) soll als eine *Menge* von ToDoListen implementiert werden, wobei diese Menge nach der Bezeichnung der ToDo-Listen sortiert ist.

Einer **`ToDoList`** kann ein neues `toDo` am Ende der Liste (`addToDo(toDo)`) oder an einer bestimmten Position (`addToDo(toDo,position)`) hinzugefügt werden - aber nur dann, wenn es das `toDo` in der Liste noch nicht gibt. ToDo-Listen mit dem gleichen Namen kann es dagegen geben. Wenn ein `toDo` einer `ToDoList` erfolgreich hinzugefügt wurde, wird `true` zurückgegeben, anderenfalls `false`. Das Entfernen eines toDos (`remove(toDo)`) gibt die Position des entfernten `todo`s zurück. Im Fall, dass es das toDo in der Liste nicht gibt, wird `-1` zurückgegeben. Die Methode `getNumberOfToDos()` gibt die Anzahl von `toDo`s in der Liste zurück. Die Klasse `ToDoListImpl` hat zusätzlich einen Konstruktor, der eine „leere“ Liste mit einer Bezeichnung (`name`) anlegt. Des Weiteren hat diese Klasse einen getter und setter.

Eine **`ToDoListCollection`** kann eine neue `ToDoList` aufnehmen (`add(list)`) und zwar in sortierter Reihenfolge nach der Bezeichnung der `ToDoList`. Die Methode `remove(list)` entfernt eine `ToDoList` aus der Kollektion. Für beide Methoden `add()` und `remove()` gilt: wenn die Methode erfolgreich ausgeführt wurde, gibt sie `true` zurück, anderenfalls `false`. Die Methode `getNumberOfToDoLists()` gibt die Anzahl von `ToDoList`s in der Liste zurück.

Teilaufgabe 4.1:

Implementieren Sie die beiden Klassen des Entwurfsklassendiagramms in Java!

- **Achten Sie im Code darauf, die gleichen Namen zu verwenden (Konsistenz von Diagramm und Code)! Die Interfaces sind gegeben.**

```
public interface ToDoListCollection{
    public boolean add(ToDoList list);
    public boolean remove(ToDoList list);
    public int getNumberOfToDoLists();
}

public interface ToDoList{
    public boolean addToDo(String toDo);
    public boolean addToDo(String toDo, int position);
    public int remove(String toDo);
    public int getNumberOfToDos();
}

import java.util.*;
```

ToDoListCollectionImpl

```
import java.util.*;
✍
                    ToDoListImpl
```

Teilaufgabe 4.2:

Um welches Interface muss das Entwurfsklassendiagramm ergänzt werden, damit die ToDo-Listen in `ToDoListCollectionImpl` nach der Bezeichnung (`listName`) der ToDo-Listen (`ToDoListImpl`) sortiert werden können? Zeichnen Sie das notwendige Interface und die entsprechende Methode in das Diagramm ein!

- **Tipp: Denken Sie auch in Ihrer Implementierung an diese Ergänzungen!**

Teilaufgabe 4.3:

Welche(s) Entwurfsmuster sind (ist) im Entwurf enthalten? Zeichnen Sie diese(s) als Kollaboration (UML-Notation) in das Diagramm ein!