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 2013

Name:	
Vorname:	
Immatrikulationsnummer:	

Aufgabe	Maximale Punktanzahl	Erreichte Punktanzahl
Teil I	45	
Teil II	45	
Gesamt	**90**	

Zum Bestehen der Klausur müssen mindestens 20 Punkte im Teil I (Analyse) und ebenfalls 20 Punkte im Teil II (Entwurf und Implementierung) erreicht werden.

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**!

Teil I (Analyse)

Aufgabe I-1: Sparschwein (13 Punkte)

Ein `Sparschwein` hat einen Inhalt (`inhalt`), ist aber zu Beginn leer und kann solange mit einzelnen Eurostücken aufgefüllt werden (`fuellen()`), bis es voll ist. Es passen genau 100 Eurostücke in das Sparschwein. Solange das Sparschwein nicht leer ist, können jederzeit wieder einzelne Euros entnommen werden (`leeren()`). Solange das Sparschwein nicht voll ist, kann jederzeit wieder ein Euro eingezahlt werden.

Lösen Sie folgende Teilaufgaben:

a) **Modellieren Sie das Sparschwein in einem UML-Analyseklassendiagramm! Denken Sie auch an die möglichen Zustandswerte!**
b) **Modellieren Sie mit Hilfe eines UML-Zustandsdiagramms das Sparschwein als Protokollmaschine!**

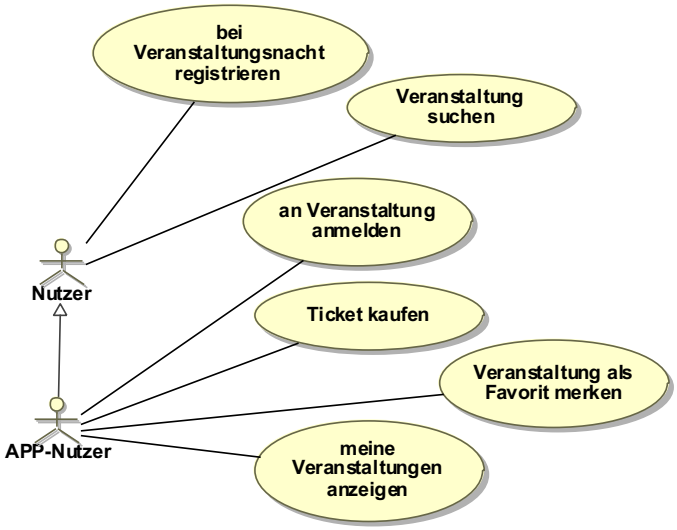
Beachten Sie folgende Hinweise:

- **Verwenden Sie die in der textuellen Beschreibung des Sparschweins gekennzeichneten Begriffe in Ihren Modellen wieder!**
- **Achten Sie auf Konsistenz beider Modelle!**
- **Es sind alle prinzipiell möglichen Zustandsübergänge zu berücksichtigen.**
- **Das Sparschwein soll unendlich lange leben.**

<u>Aufgabe I-2:</u> VeranstaltungsAPP (32 Punkte)

Veranstaltungsnächte werden vielerorts zu verschiedenen Themen organisiert, so z.B. die Lange Nacht der Wissenschaften in Dresden. Ein Softwareunternehmen bekommt die Aufgabe, dazu eine Software für mobile Endgeräte (APP) für die Besucher der Veranstaltungsnacht zu entwickeln. Dazu wird zunächst eine Anforderungsanalyse gemacht und aus der daraus resultierenden Beschreibung ein Domänenmodell entwickelt. Die Beschreibung ist die folgende:

Eine VeranstaltungsAPP liefert Informationen für die Veranstaltungen einer Veranstaltungsnacht. Eine Veranstaltungsnacht findet zu einem bestimmten Datum statt. Ein Nutzer kann sich bei der APP mit seinem Namen registrieren, kann diese aber auch anonym (und dann eingeschränkt) nutzen. Nutzer sind genau dann APP-Nutzer, wenn sie sich bei der Veranstaltungsnacht registriert haben. Das folgende Anwendungsfalldiagramm zeigt die Anwendungsfälle für die VeranstaltungsAPP.



Die Veranstaltungen einer Veranstaltungsnacht sind in einem Veranstaltungskatalog zusammengefasst. Ein Nutzer kann in diesem Veranstaltungskatalog Veranstaltungen suchen. Falls er registrierter Nutzer (APP-Nutzer) ist, kann er sich „seine“ Veranstaltungen, für die er sich interessiert, als Favoriten merken. Außerdem kann er sich für die gesamte Veranstaltungsnacht ein oder mehrere Tickets kaufen, mit denen er kostenpflichtige Services der Veranstaltungsnacht, wie z.B. kostenpflichtige Veranstaltungen, für sich und seine begleitenden Personen nutzen kann.

Für einige Veranstaltungen (egal ob kostenpflichtig oder nicht) muss man sich als registrierter Nutzer anmelden (sich und ggfs. weitere Personen). Da eine Veranstaltung typischerweise in einer Veranstaltungsnacht wiederholt durchgeführt wird, bekommt jede(r) Veranstaltung(styp) deren Uhrzeiten (Startzeit und wenn relevant die Endzeit der Veranstaltung) zugeordnet. Eine Anmeldung bedeutet, dass zunächst und wenn notwendig (vom Server der Veranstaltungsnacht) geprüft werden muss, ob es noch die (gesamte) Anzahl der gewünschten freien Plätze gibt. Entsprechend erhält die Anmeldung einen Status: *angefordert*, *reserviert* oder *auf Warteliste*.

Ein wichtiger Zweck der APP ist die Information über alle Veranstaltungen der Veranstaltungsnacht. Eine Veranstaltung wird beschrieben durch

- einen Titel sowie eine kurze Beschreibung (Abstract) der Veranstaltung
- die Information, ob die Veranstaltung kostenfrei und anmeldepflichtig ist
- Informationen über den Veranstaltungsort: Adresse, optional einen Raum und eine Information über die Barrierefreiheit
- optional Verweise auf Medien (wie z.B. Bilder), die einen Titel tragen
- Kategorie mit Merkmalen einer Veranstaltung: Ausstellung mit Exponaten, Show mit der Anzahl von Plätzen, Präsentation mit Vortragenden
- Optional Links zu Einträgen in sozialen Netzwerken (mit dem Namen der Community und der URL)

Jeder Veranstaltungsort bekommt über seine GPS-Koordinaten eine Visualisierung auf einer Veranstaltungskarte angezeigt.

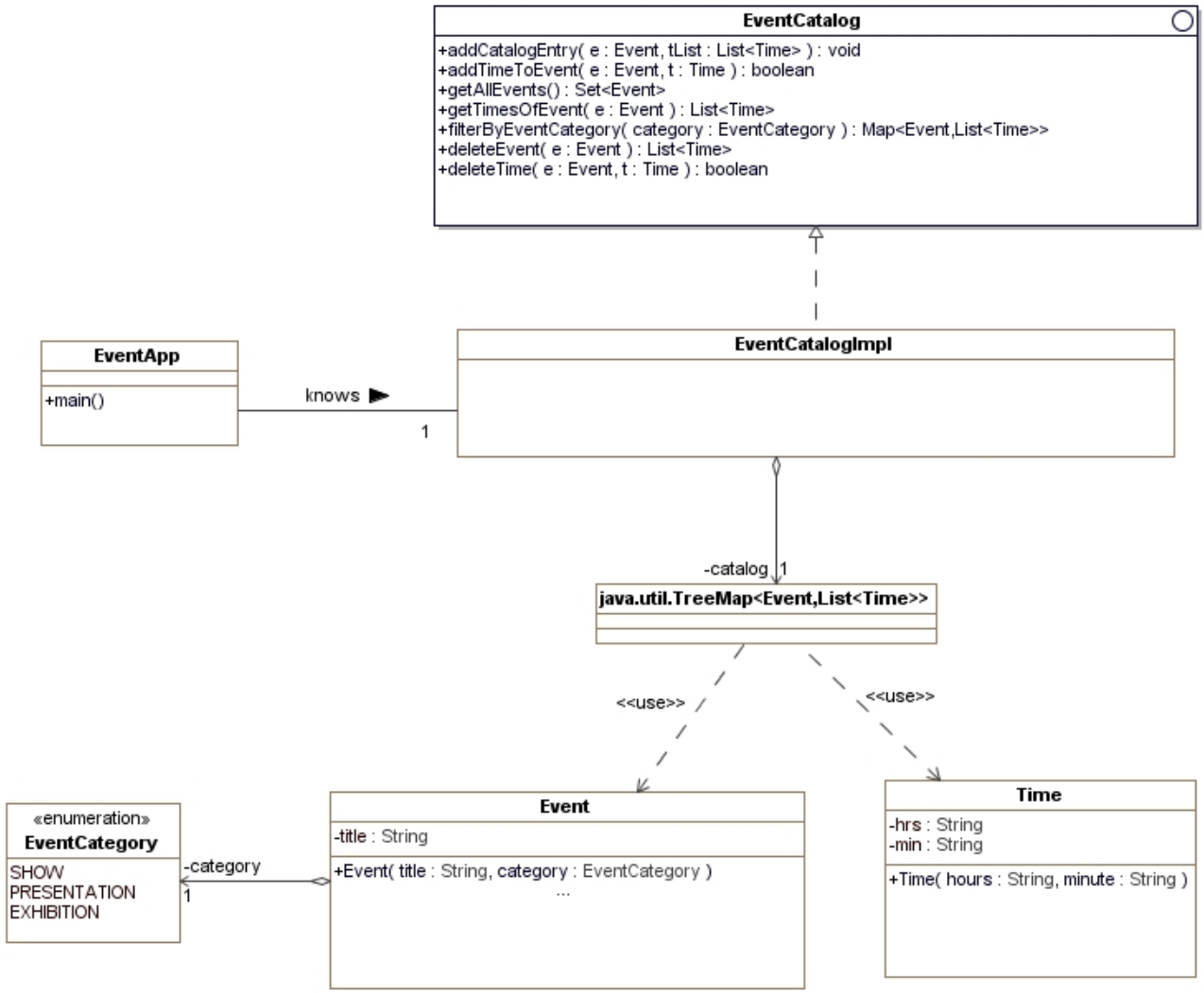
Modellieren Sie VeranstaltungsAPPs mit Hilfe eines UML-Analyseklassendiagramms (Domänenmodell)!

Teil II (Entwurf und Implementierung)

Suchen im Veranstaltungskatalog (VeranstaltungsAPP)

Nach der Analyse wird in einem ersten Prototyp (mit reduzierter Funktionalität) der Veranstaltungskatalog (`EventCatalog`) implementiert. Dazu verfeinert ein Softwareentwickler sein Domänenmodell zu einem Entwurfsmodell in folgender Weise. Er wechselt auch in der Namensbildung von deutsch auf englisch. Das Interface `EventCatalog` definiert alle Methoden für einen Veranstaltungskatalog. Die Klasse `EventCatalogImpl` implementiert das Interface:

- `addCatalogEntry()` fügt einen Katalogeintrag bestehend aus einer Veranstaltung (`Event`) und einer Liste von Uhrzeiten (`Time`) dieser Veranstaltung dem Katalog hinzu.
- `addTimeToEvent()` fügt einem Katalogeintrag eine Uhrzeit hinzu.
- `getAllEvents()` liefert alle Veranstaltungen (ohne Uhrzeiten).
- `getTimesOfEvent()` liefert alle Uhrzeiten einer Veranstaltung.
- `filterByEventCategory()` liefert alle Katalogeinträge einer ausgewählten Veranstaltungskategorie (`EventCategory`).
- `deleteEvent()` löscht eine Veranstaltung einschließlich deren Uhrzeiten im Katalog.
- `deleteTime()` löscht eine Uhrzeit einer Veranstaltung.



Lösen Sie folgende Teilaufgaben:

(1) **Implementieren Sie die Klasse `EventCatalogImpl`! (22 Punkte)**

```
import java.util.*;

public class EventCatalogImpl ✎                                    {
✎
```

```
}
```

(2) Tragen Sie Modellelemente (Methoden, Interfaces, Klassen?), die Ihnen für die Implementierung der Klasse fehlen, in das Entwurfsdiagramm ein! (3 Punkte)

(3) Die Implementierung des Veranstaltungskataloges erfolgt mit einem Objektadapter. Zeichnen Sie diesen Objektadapter in UML-Notation in das obige Klassendiagramm ein. (3 Punkte)

(4) Ändern Sie in einem neuen Prototyp die Implementierung von dem Objektadapter auf einen Klassenadapter! (6 Punkte)

- **Beschreiben Sie kurz, was sich im Entwurfsmodell ändert!**

- **Implementieren Sie dazu die notwendigen Änderungen im Code der Klasse `EventCatalogImpl` und beispielhaft an der Methode `addTimeToEvent()`!**

(5) Erstellen Sie für die Ausführung der folgenden `main()`-Methode der Klasse `EventAPP` ein Sequenzdiagramm! Parameter und Rückkehrwerte sollen dabei nicht berücksichtigt werden. Gehen Sie bei der Implementierung der Klasse `EventCatalogImpl` von der Version des Objektadapters aus! (11 Punkte)

```
public static void main(String[] args) {
  EventCatalogImpl catalog = new EventCatalogImpl();
  Event show1 = new Event("show1",EventCategory.SHOW);
  catalog.addCatalogEntry(show1, new ArrayList<Time>());
  Event show2 = new Event("show2",EventCategory.SHOW);
  catalog.addCatalogEntry(show2, new ArrayList<Time>());
  catalog.addTimeToEvent(show1, new Time("18","00"));
  catalog.deleteEvent(show2);
  Set<Event> allEvents = catalog.getAllEvents();
  Map<Event,List<Time>> allShows =
                        catalog.filterByEventCategory(EventCategory.SHOW);
}
```