Examination
Software Engineering (SE)

Prof. Dr.-Ing. Christian Heller <christian.heller@ba-leipzig.de>

Personal Data	
First and Last Name	Ihre Daten müssen von der Klausuraufsichtsperson schriftlich auf der Anwesenheitsliste festgehalten werden!
Matriculation Number	
Subject and Year	CS 2013
Login	Das Login "klaus..." muss unbedingt schriftlich auf der Anwesenheitsliste festgehalten werden, da sonst keine Zuordnung des Logins zu Ihrem Namen und damit keine Korrektur der Klausur möglich ist!

Examination Data	
Date	2015-09-28
Duration [min]	120
Maximum Points [Credit Point]	100
Permitted Study Aids	Dokumentation im lokalen Netzwerkverzeichnis (Intranet); NICHT gestattet sind Kommunikationsmöglichkeiten (Internet) oder Anmeldung via SSH auf dem Rechner "fileserv", wo Ihr Homeverzeichnis liegt, oder eine Anmeldung mit Ihrem Klausur-Login nach Ende der Prüfung. Dies kann leicht geprüft werden (last cs12*, Server-Log-Dateien). Bitte unterlassen Sie also Täuschungsversuche in Ihrem eigenen Interesse.
Remarks	Hinterlegen Sie alle Programme und Antworten in elektronischer Form! Es wird kein Papier angenommen. Möchten Sie Lösungen erläutern, so nutzen Sie Quelltext-Kommentare oder legen eine Text-Datei an. Speichern Sie sämtliche Daten im HOME-Verzeichnis des Nutzers, d.h. unter Windows auf Laufwerk H:\ (NICHT auf C:\ oder "Eigene Dateien")! Idealerweise legen Sie dort ein Unterverzeichnis namens "klausur" an. Lesen Sie die Aufgaben komplett durch, bevor Sie sie lösen! Die Reihenfolge der Lösung ist Ihnen überlassen. Probieren Sie immer, eine Aufgabe zu lösen, da auch auf richtige Teile nicht vollständiger Lösungen Punkte vergeben werden! Falls vom Prinzip her richtig, so werden auch alternative Lösungen akzeptiert. Sie dürfen beliebig viele Bildschirmausgaben von Werten in den Quelltext einbauen, um ein Programm besser nachvollziehen zu können. Diese Aufgabenstellung in Papierform können Sie nach dem Ende der Klausur behalten.

Evaluation											
Task	1	2	3	4	5	6	7	8	9	10	Summe
Desired Value [Credit Point]	20	20	20	10	20	10	0	0	0	0	100
Actual Value [Credit Point]											

Im Rahmen der Klausur sind sechs Aufgaben verschiedener Themen, immer aber mit Bezug zur *Unified Modeling Language* (UML) zu lösen. Einige der Aufgaben sind der Anwendungsdomäne *Autovermietung* entnommen. Die Aufgaben 2 und 3 hängen zum Teil von Ergebnissen aus Aufgabe 1 ab. Ansonsten sind die Aufgaben autonom.

Insofern modelliert werden soll, ist das *BOUML*-Werkzeug zu verwenden.

Task 1: Structure [20]

In dieser Aufgabe sollen die wichtigsten Fachklassen und Beziehungen einer Kfz-Vermietung näher untersucht und in einem geeigneten Diagramm modelliert werden. Kardinalitäten sind stets für *beide* Richtungen anzugeben. Die Richtung der Beziehungen (Pfeile) kann vernachlässigt werden.

a) Es gibt Kraftfahrzeuge (*Kfz*). Jedes Kfz gehört zu genau einem *KfzTyp*; zu einem KfzTyp wird es beliebig viele Kfz geben können. [4]

b) *Reservierungen* beziehen sich immer auf genau einen KfzTyp (und nicht auf ein konkretes Kfz). *Vermietungen* hingegen beziehen sich immer auf ein konkretes Kfz. [4]

c) Zu einer Vermietung kann ein *Rücknahmeprotokoll* geschrieben werden. [2]

d) Vor einer Vermietung kann ein Kfz reserviert worden sein. Umgekehrt kann es passieren, dass trotz Reservierung kein Kfz für eine Vermietung zur Verfügung steht. [2]

e) Sowohl Reservierung, als auch Vermietung können in beliebiger Anzahl durch einen *Mieter* veranlasst werden. [4]

f) Der Mieter (z.B. eine Firma) ist jedoch nicht zugleich auch der Fahrer. Daher bestimmt ein Mieter eine Reihe von *KundenMitarbeitern*, die als mögliche Fahrer in Frage kommen. Diese müssen sich jedoch selbstständig mit ihren Personalien für eine Vermietung registrieren. Eine Vermietung ohne Fahrer ist nicht möglich. [4]

Task 2: Physical Architecture [20]

Ziel dieser Aufgabe ist die Darstellung einer so genannten *Information Technology* (IT)-Landschaft einer Autovermietungsfirma mittels Verteilungsdiagrammes. Außerdem sollen Quelltext und Dokumentation generiert werden.

a) Es gibt eine zentrale Datenhaltung auf einem *Application-Server*. Er verwendet intern einen *Enterprise Java Beans* (EJB) *Container* und ein *Oracle Datenbank-Management-System* (DBMS). Beide sind via *Java Database Connectivity* (JDBC) miteinander verbunden. [4]

b) Mit dem *Application-Server* über *Remote Method Invocation/ Local Area Network* (RMI/LAN) verbunden ist ein *Web-Server*. [2]

c) Vier verschiedene Benutzerkreise greifen über *Clients* via *http* auf den *Web-Server* zu: die *Kunden*, die über das *Internet* reservieren können; das interne *Callcenter*, welches eine umfassendere Anwendung bekommt, die über das firmeninterne *LAN* läuft sowie *Agenturen* und *Stationen*, die der Einfachheit halber ebenfalls über das *Internet* angebunden sind. [4]

d) Fügen Sie einen *Artefakt* namens *Quelltext* hinzu, welcher durch eine lose Beziehung mit dem *Application-Server* verbunden ist! Ordnen Sie ihm sämtliche Klassen aus Aufgabe 1 zu! [4]

e) Generieren Sie den zum Artefakt gehörigen Java-Quelltext in ein Unterverzeichnis *src*! [4]

f) Hinterlegen Sie zum Verteilungsdiagramm beispielhaft eine Beschreibung (ein Satz genügt) und generieren Sie eine Dokumentation (nur zum Verteilungsdiagramm) im HTML-Format in ein Unterverzeichnis namens *doc*! [2]

Task 3: Logical Architecture [20]

In Aufgabe 1 erstellte Klassen sollen Komponenten zugeordnet und diese verbunden werden.

a) Fassen Sie Klassen (als „provided interface“) in Komponenten zusammen, wie folgt [5]:

- Komponente *Kunde*: Mieter, KundenMitarbeiter
- Komponente *Vermietung*: Reservierung, Vermietung, RücknahmeProtokoll
- Komponente *Kfz*: Kfz, KfzTyp

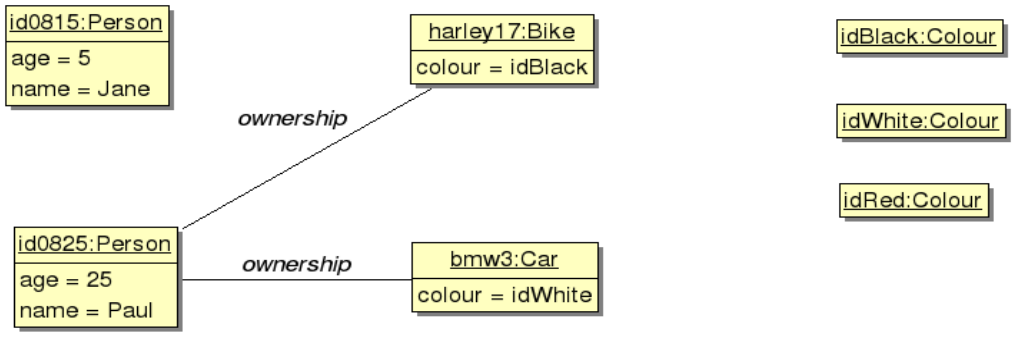
b) Verdeutlichen Sie über eine Schnittstelle, dass ein KundenMitarbeiter als Fahrer nötig ist, um ein Auto zu mieten! [5]

c) Verdeutlichen Sie über zwei weitere Schnittstellen, dass:

- ein Kfz nötig ist, um ein Auto zu mieten [5];
- ein KfzTyp nötig ist, um eine Reservierung durchzuführen [5]!

Task 4: Object Constraint Language (OCL) [10]

Gegeben sei das folgende Objektdiagramm:



Treffen Sie eine Aussage, ob die folgenden Objektabfragen *wahr* oder *falsch* sind:

a) [2]

context Vehicle

inv: self.owner.age >= 18

b) [2]

context Person

inv: self.fleet–>forAll(v | v.colour = #black)

c) [2]

context Person

inv: self.fleet–>select(v | v.colour = #black)–>size <= 3

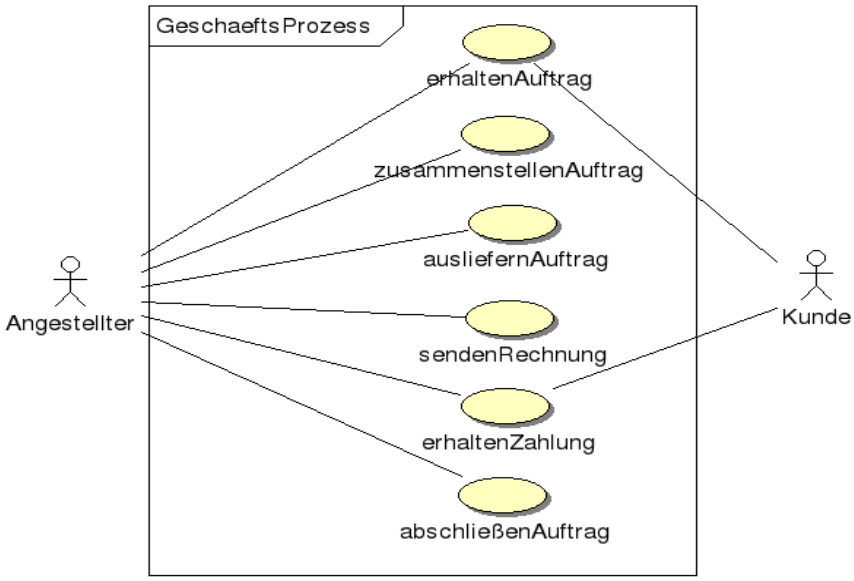
d) [2]

inv: Car.allInstances()–>exists(c | c.colour=#red)

e) Welche Art von Bedingungen kann man mittels der OCL im Zusammenhang mit Operationen angeben? [2]

Task 5: Behaviour [20]

Ziel dieser Aufgabe ist die Modellierung eines Geschäftsprozesses bzw. Arbeitsflusses. Zum besseren Verständnis ist ein Teil seiner Vorgänge im folgenden Anwendungsfalldiagramm dargestellt. Nutzen Sie das UML-Aktivitätsdiagramm, um den beschriebenen Geschäftsprozess abzubilden!



a) Ein Sachbearbeiter nimmt einen Auftrag entgegen. [1]

b) Als nächstes muss er zum Einen den Auftrag zusammenstellen und auch ausliefern. [2]

c) Desweiteren muss er eine Rechnung senden sowie die Zahlung entgegennehmen. [2]

d) Da die Logistik hin-und-wieder klemmt, springt Otto oft zwischen den unter b) und c) genannten Vorgangssträngen hin und her, bearbeitet sie also quasi parallel. [2]

e) Am Ende schließt Otto den Auftrag ordnungsgemäß ab, womit der Geschäftsfall endet. [1]

f) Ordnen Sie die Aktionen Objekten zu, wie folgt [2]:

- Objekt *Lager*: zusammenstellenAuftrag, ausliefernAuftrag
- Objekt *Kundenbetreuung*: erhaltenAuftrag, sendenRechnung, abschlieszenAuftrag
- Objekt *Buchhaltung*: erhaltenZahlung

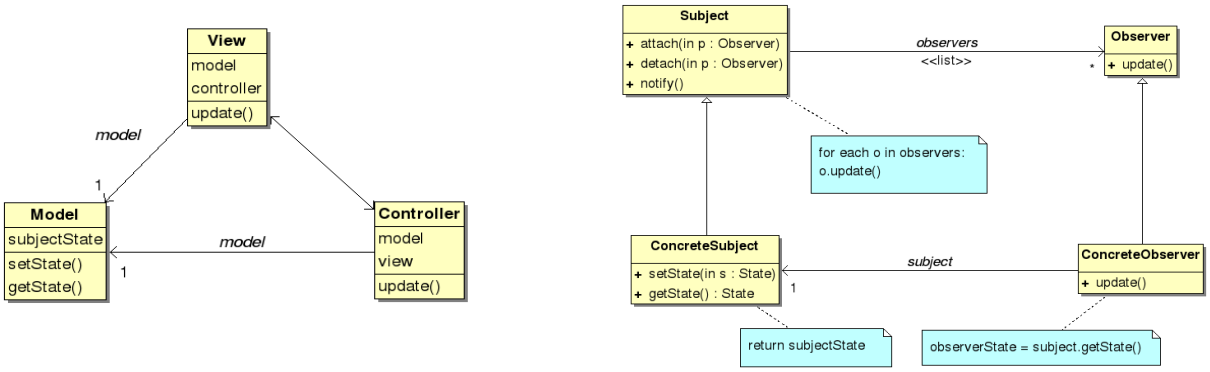
g) Das (in b) bereits erwähnte) Ausliefern besteht aus Teilaktivitäten, die als Verfeinerung in einem eigenen Diagramm darzustellen sind. [2]

h) Eine Aktivität namens "ausliefernAuftrag" soll alle Elemente des zu zeichnenden Diagrammes beinhalten. Sie soll außerdem einen Ein- sowie einen Ausgabeparameter, beide mit dem Namen "auftrag" enthalten. [2]

i) Handelt es sich um einen Eilauftrag, so soll per Express ausgeliefert werden; anderenfalls per normaler Post. [6]

Task 6: Software Pattern [10]

Gegeben seien die beiden Softwaremuster "Model-View-Controller" und "Observer":



Kombinieren Sie beide in sinnvoller Weise und modellieren Sie das resultierende Muster unter Verwendung des UML-Klassendiagrammes!

Hinweise: Die gegebenen Klassennamen können geändert werden. Die Kommentarelemente brauchen nicht mit gezeichnet zu werden. Lassen Sie sich nicht durch Attribute oder Methoden irritieren, sondern konzentrieren Sie sich auf das Einbetten des einen- in das andere Muster!

Zusatzaufgabe: Nennen Sie Frameworks, in denen das Muster in dieser Form oder Varianten davon vorkommt! Wie heißen die Varianten bzw. Synonyme? [2]

Viel Erfolg!