Prüfung
Software Engineering (SE)

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

Student	
Vor- und Nachname	Ihre Daten werden von der Klausuraufsichtsperson schriftlich auf der Anwesenheitsliste festgehalten.
Matrikelnummer	
Studienrichtung und Jahr	CS 2017-2
Anmeldename	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!

Prüfung	
Datum	September 2019
Dauer [min]	100
Hilfsmittel	Dokumentation im lokalen Netzwerk (Intranet) sowie Recherche im Internet. NICHT gestattet: * Kommunikation in jeglicher Form * Anmeldung via SSH auf dem Rechner "fileserv" * Anmeldung mit Klausur-Login nach Ende der Prüfung Dies kann leicht geprüft werden (last cs16*, Server-Log-Dateien). Bitte unterlassen Sie also Täuschungsversuche in Ihrem eigenen Interesse.
Bemerkungen	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. Bitte duplizieren Sie Ihre Quelltextdateien (workspace) NICHT, da beim Korrigieren dann beide durchsucht werden müssen, was sinnlosen Aufwand verursacht. Diese Aufgabenstellung in Papierform können Sie nach dem Ende der Klausur behalten.

Bewertung						
Aufgabe	1	2	3	4	5	Summe
Punkte	20	20	20	20	20	100

Im Rahmen der Klausur sind Aufgaben mit Bezug zur Unified Modeling Language (UML) zu lösen. Insofern modelliert werden soll, ist das BOUML-Werkzeug zu verwenden.

Aufgabe 1: Anwendungsfall [20]

Zweck: Formulieren allgemeiner Anwendungsfälle (Use Cases) aus konkreten Szenarien.

Beschreibung: Ihre Kollegin hat drei Szenarien für den Vorgang „Reservieren eines Hotelzimmers“ gesammelt. Sie wurden bereits überarbeitet, so dass sie sich sehr ähneln.

Szenario Herr Schmidt:

1. Herr Schmidt reserviert ein Zimmer via Buchungs-App seines Funktelefones.
2. Das System zeigt verfügbare Zimmerkategorien und Preise an.
3. Herr Schmidt wählt die Standardkategorie aus und gibt an, den ganzen Monat April bleiben zu wollen.
4. Das System berechnet die Kosten auf 29 x 120 EUR = 3.480 EUR.
5. Herr Schmidt reserviert für sich ein Zimmer der Standardkategorie.

6. Das System bezieht die tatsächlich zur Verfügung stehenden Zimmer der Standardkategorie von der Datenbank.
7. Das System erstellt eine neue Reservierung für ein Standardzimmer.
8. Das System zeigt die Bestätigungsnummer 4711 für die Reservierung an und gibt Erläuterungen zur Anreise.

Szenario Frau Ahorn:

1. Frau Ahorn entscheidet sich, per öffentlichem Terminal in ihrer lokalen Postfiliale ein Zimmer zu reservieren.
2. Das System zeigt verfügbare Zimmer und Preise des gewünschten Hotels an.
3. Frau Ahorn wählt die Damen-Suite und gibt als Eckdaten für ihren Aufenthalt 31. Oktober bis 1. November ein.
4. Das System berechnet die Kosten auf 1 x 275 EUR = 275 EUR.
5. Frau Ahorn führt die Reservierung der Damen-Suite durch.
6. Das System markiert die Damen-Suite in der Datenbank als blockiert.
7. Das System erzeugt eine Reservierung für die Damen-Suite vom 31. Oktober bis zum 1. November.
8. Das System zeigt die Bestätigungsnummer 4712 für die Reservierung an und gibt Erläuterungen zur Anreise.

Szenario Johannes:

1. Im Rechenzentrum seiner Universität reserviert Johannes an einem Desktop-Rechner ein Zimmer.
2. Das System zeigt verfügbare Zimmerkategorien und Preise eines Hotels an.
3. Johannes wählt ein Billigzimmer aus und gibt an, in der ersten Juli-Woche bleiben zu wollen.
4. Das System berechnet die Kosten als 6 x 40 EUR = 240 EUR.
5. Johannes schließt die Reservierung für das preisgünstige Zimmer ab.
6. Das System bezieht die Anzahl verfügbarer Billigzimmer von der Datenbank.
7. Das System erstellt eine Reservierung für ein Billigzimmer für die erste Juli-Woche.
8. Das System zeigt die Bestätigungsnummer 4713 für die Reservierung an und gibt Hinweise zur Anreise.

Abstrahieren Sie den Fluss der Aktionen für den Anwendungsfall „Reservieren Raum“ und schreiben die resultierenden Schritte auf! [3 Punkte je Schritt = 21 Punkte (davon 1 Zusatz)]

Beispiel:

1. Der Kunde startet das Programm zur Zimmerreservierung.
2. ...

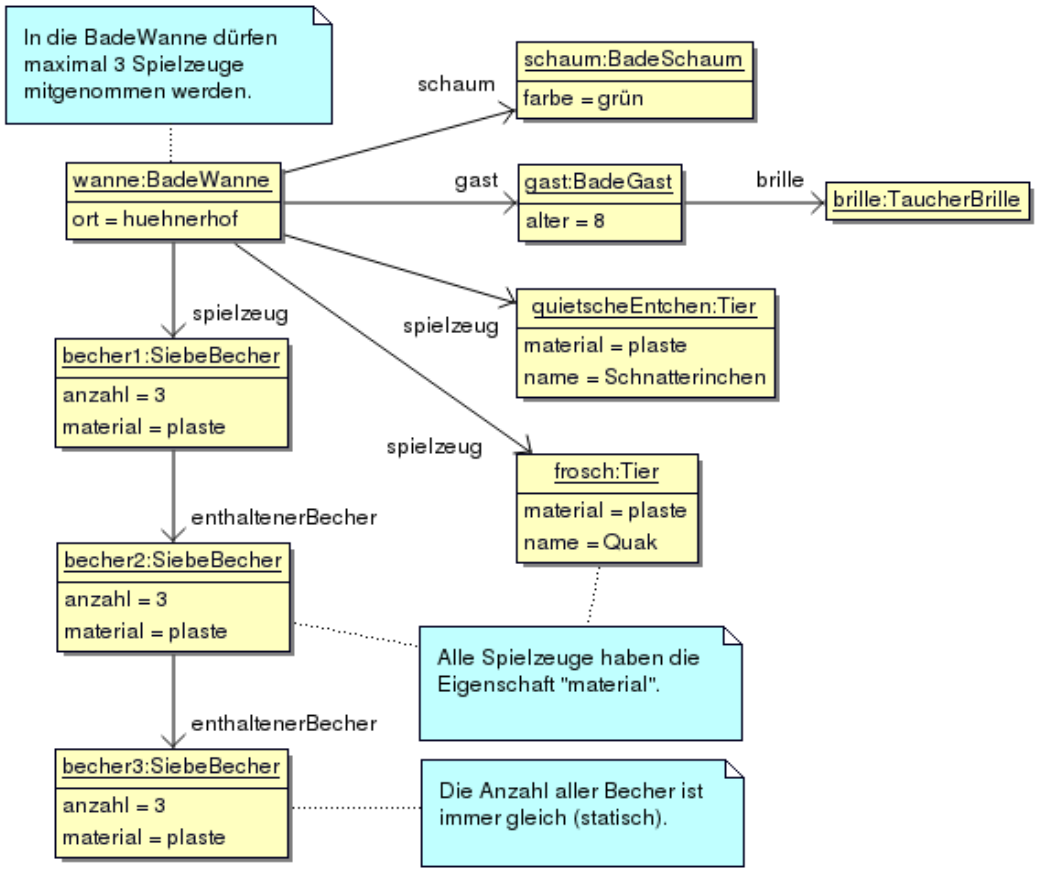
Hinweis: Schreiben Sie die Lösungen in eine Textdatei namens „aufgabe_1.txt“!

Ergebnis: Ein allgemeiner Ablauf mit allen notwendigen Schritten wurde definiert.

Aufgabe 2: Objekt [20]

Zweck: Abstrahieren von Laufzeitinstanzen eines Objektdiagramms in einem Klassendiagramm.

Erstellen Sie zu dem hier gegebenen Objektdiagramm ein passendes Klassendiagramm!



a) Klasse zum Objekt „wanne“ [2]

b) Klasse zum Objekt „schaum“ [2]

c) Klasse zum Objekt „gast“ [2]

d) Klasse zum Objekt „brille“ [2]

e) Klasse zu den Objekten „quietscheEntchen“ und „frosch“ [2]

f) Klasse zu den Objekten „becher*“ [2]

g) Superklasse mit Attribut [2]

h) Klassenattribut [2]

i) Assoziationen [2]

Hinweis: Verwenden Sie unidirektionale Beziehungen. Es reicht die Angabe der Zielrolle, ohne Multiplizität.

j) Multiplizität für Anzahl der Spielzeuge [2]

Ergebnis: Das zum Objektdiagramm gehörige Klassengerüst wurde korrekt erstellt.

Aufgabe 3: Paket [20]

Zweck: Modellieren einer logischen Schichtenarchitektur einer Anwendung im Paketdiagramm.

Hinweise: Da in BOUML kein Paketdiagramm existiert, kann ein beliebiges passendes anderes verwendet werden, um Pakete darzustellen. Auch hat BOUML keine „White Box View“ für Pakete. Daher sind Schachtelungen grafisch selbst abzubilden, indem auch innere Pakete extra mit eingezeichnet werden.

a) Die „presentation_layer“ enthält ein „user_interface“ und die „presentation_logic“. [2]

b) Die „business_layer“ umfasst den „business_workflow“, die „business_components“ und die „business_entities“. [2]

c) Die „data_layer“ besteht aus den Paketen „data_access“ und „service_agents“. [2]

d) Die „business_layer“ greift auf die „data_layer“ zu, was durch eine Abhängigkeitsbeziehung kenntlich zu machen ist. Dies geschieht in einer Weise, die die Weiterverwendung von Daten außerhalb der „business_layer“ NICHT gestattet. [2]

Hinweis: Verwenden eines passenden Stereotypes.

e) Im Gegensatz dazu verwendet die „presentation_layer“ Daten der „business_layer“ in einer Weise, die eine Weiterverwendung auch in anderen Paketen ermöglicht. [2]

f) In der „business_layer“ gibt es desweiteren ein Paket namens „application_facade“. Es hängt ab von den anderen drei Paketen der „business_layer“. Das Stereotyp ist wiederum so zu setzen, dass nur die Fassade Zugriff hat. [2]

g) Alle bisher genannten Pakete gehören zu „layered_application“. Hinweis: Schachtelung. [2]

h) Markieren Sie das Paket „layered_application“ mittels Notizzettelelement mit Anker! Schreiben Sie als Bemerkung hinein, dass es sich um das Software-Muster „Schichtenarchitektur“ handelt! [2]

i) Notieren Sie auf gleiche Weise für das Paket „data_access“, dass es sich um „Object Relational Mapping“ (ORM) handelt! [2]

j) Passen Sie das Diagramm so an, dass Paketnamen in der Karteireiterlasche erscheinen! Stellen Sie die Hintergrundfarbe für Pakete auf hellgrün ein! [2]

Ergebnis: Die logischen Schichten der Anwendung wurden korrekt dargestellt.

Aufgabe 4: Kontrollfluss [20]

Zweck: Darstellen des „Bubble Sort“-Algorithmus’ im Aktivitätsdiagramm.

Gegeben sei die Methode „sort“, welche ein Feld von Variablen des Types „int“ ordnet:

```
public int[] sort(int[] list) {
    boolean change = true;
    if (list.length > 1) {
        while (change) {
            change = false;
            for (int i = list.length - 1; i > 0; i--) {
                int i1 = list[i];
                int i2 = list[i - 1];
                if (i1 < i2) {
                    list[i] = i2;
                    list[i - 1] = i1;
                    change = true;
                }
            }
        }
    }
    return list;
}
```

a) Bilden Sie den Kontrollfluss des Algorithmus' per Aktivitätsdiagramm ab! [2]

b) Kapseln Sie den Inhalt des Diagrammes grafisch durch eine Aktivität „sort“, welche den Eingabeparameter „list“ und einen Rückgabewert „result“ anzeigt! [2]

Hinweis: Der Kontrollfluss soll vom Parameter ausgehen und mit dem Rückgabewert enden.

c) Beginnen Sie mit der Aktion „setChange“! [2]

Hinweis: Sie dient der Initialisierung der Bool’schen Variable „change“.

d) Bauen Sie als nächstes eine Verzweigung ein, welche das erste „if“ repräsentiert! Geben Sie, wo nötig, Bedingungen an! [2]

e) Verwenden Sie eine weitere Verzweigung, um die „while“-Schleife nachzubilden! [2]

f) Kapseln Sie die Zuweisung eines Variablenwertes in einer Aktion „resetChange“! [2]

g) Fügen Sie die Verzweigung für die „for“-Schleife ein! [2]

h) Kapseln Sie die Zuweisung von Variablenwerten in einer Aktion „assignVariables“! Geben Sie konkrete Inhalte der Zuweisungen per Notizzettelelement an! [2]

i) Fügen Sie schließlich eine letzte Verzweigung ein, welche das „if (i1 < i2)“ repräsentiert! [2]

j) Kapseln Sie nachfolgende Zuweisungen in den Aktionen „swapValues“ und „setChangeAgain“, wobei Details wiederum per Notizzettelelement anzugeben sind! [2]

Hinweis: Prüfen Sie abschließend nochmals alle Transitionen! Beachten Sie, dass überall, wo nötig, Bedingungen angegeben wurden!

Ergebnis: Das Aktivitätsdiagramm stellt Parameter, Aktionen, Verzweigungen und Schleifen dar.

Aufgabe 5: Muster [20]

Zweck: Modellieren des „Model View Controller“ (MVC) Software-Musters im Klassendiagramm.

Beschreibung: Das klassische MVC enthält das „Beobachter“-Muster.

Erstellen Sie ein Klassendiagramm, in welchem alle Bestandteile des MVC enthalten sind! Machen Sie durch Beschriftung kenntlich, welche Teile zum „Beobachter“-Muster gehören!

a) Klassen des MVC [2]

b) Abstrakte Superklasse „Subject“ von Model [2]

c) Super-Schnittstelle „Observer“ von View und Controller [2]

d) Assoziation namens „update“ zwischen Subject und Observer [2]

e) Assoziation namens „register / unregister“ jeweils zwischen View und Model sowie Controller und Model [2]

f) Methode „notify“ in Klasse Subject [2]

g) Methode „update“ in Observer und View [2]

h) Methode „handle“ in Controller [2]

i) Attribut „model“ in View und Controller [2]

j) Attribut „views“ in Controller als Liste [2]

Ergebnis: Das MVC-Muster wurde vollständig und korrekt dargestellt.

Viel Erfolg!