Aufbewahrung bis: 2025-07-27

Unterweisung zu Verhaltensrichtlinien

Prüfung	inf031, Objektorientierte Modellierung und Programmierung
Datum	2020-07-27
Uhrzeit	17:00
Raum	Hörsaal 1

- Bitte tragen Sie Ihren Mund-Nasen-Schutz, wenn Sie sich im Raum oder im Gebäude bewegen.
- Wenn Sie eine Frage haben, setzen Sie bitte Ihren Mund-Nasen-Schutz auf und geben einen kurzen mündlichen Hinweis. Eine Aufsichtsperson wird dann den Mund-Nasen-Schutz ebenfalls aufsetzen und sich Ihnen soweit es der Mindestabstand und die Bedingungen hier im Raum zulassen annähern, damit sie Rücksprache halten können.
- Wenn Sie die Toilette benutzen müssen, setzen Sie bitte Ihren Mund-Nasen-Schutz auf und geben ein Handzeichen. Eine Aufsichtsperson wird dann ggf. Personen, die mit Ihnen in der Reihe sitzen, bitten, ebenfalls ihren Mund-Nasen-Schutz aufzusetzen und Ihnen den Weg unter Einhaltung der Mindestabstände frei zu machen. Bei der Rückkehr an den Platz wird ebenso verfahren.
- Ein endgültiges Verlassen des Prüfungsraums ist erst nach Ende der Prüfung möglich.
- Nach Ende des Prüfungszeitraums werden Sie gebeten, den Mund-Nasen-Schutz aufzusetzen. Bitte räumen Sie dann zügig Ihre Sachen zusammen und folgen den Anweisungen zum Verlassen des Raumes. Ihre Klausurbögen, das aufgefüllte Formular mit Ihren Kontaktdaten sowie diese Unterweisung lassen Sie an Ihrem Platz liegen. Diese werden anschließend eingesammelt.
- Nach Verlassen des Raumes verlassen Sie bitte auch umgehend, aber ruhig und geordnet das Gebäude. Vermeiden Sie dabei Gruppenbildung auf den Fluren und an den Ausgängen.

Ich habe die Hinweise zur Kenntnis genommen.

Name in Druckbuchstaben

2020-07-27

Datum **Unterschrift**

Aufbewahrung bis: 2020-08-17

Erhebung personenbezogener Daten

Prüfung	inf031, Objektorientierte Modellierung und Programmierung
Datum	2020-07-27
Uhrzeit	17:00
Raum	Hörsaal 1

Die folgenden Daten werden im Rahmen des § 2h S. 4 Corona-VO erhoben und für drei Wochen aufbewahrt.
Bitte füllen Sie das Formular vollständig aus:

Name	
Vorname	
Vollständige Anschrift	
Telefon	

Den Datenschutzhinweis finden Sie im Stud.IP-Eintrag dieser Veranstaltung im Dateibereich.

Carl von Ossietzky
Universität
Oldenburg

Dr. C. Schönberg
Carl von Ossietzky Universität Oldenburg, Fakultät II, Department für Informatik

Klausur – Gruppe A

Objektorientierte Modellierung und Programmierung (inf031)

Sommersemester 2020

- Lassen Sie die Blätter in jedem Fall zusammengeheftet.
- Geben Sie auf dieser Seite Name, Vorname und Matrikelnummer an und unterschreiben Sie die Erklärung.
- Notieren Sie die Lösungen zu den Aufgaben direkt auf dem Aufgabenzettel oder auf der vorhergehenden oder nachfolgenden Rückseite.
  Wenn Sie Lösungen an anderer Stelle notieren, markieren Sie dies deutlich sowohl dort als auch auf der Seite der Aufgabe.
- Für die Bearbeitung der Klausur stehen Ihnen 120 Minuten zur Verfügung.
- Schalten Sie Mobiltelefone und ähnliche Geräte aus und verstauen Sie sie sicher.
  Während der Klausur sichtbare technische Hilfsmittel gelten als Täuschungsversuch.
- Benutzen Sie nur Stifte mit blauer oder schwarzer Tinte.
  Alles, was mit Bleistift oder in roter Tinte geschrieben ist, wird nicht gewertet.
- Als einziges Hilfsmittel ist ein doppelseitig *handbeschriebenes* DIN A4 Blatt zugelassen.

Name, Vorname: _____

Matrikelnummer: _____

Erklärung: Ich versichere, dass ich die Klausur selbstständig, ohne fremde Hilfe und ohne weitere Hilfsmittel bearbeitet habe.

_____
Unterschrift

Bewertung (vom Prüfer auszufüllen):

Aufgabe:	1	2	3	4	5	6	$\sum$
Punkte:	20	15	21	16	15	13	100
Erreicht:							

Aufgabe 1: *Entwurf in Java* **(10 + 1 + 9 Punkte)**

Der berühmte Archäologe *Illinois Jones* will die Artefakte, die er auf diversen Forschungsreisen entdeckt hat, katalogisieren.

Ein ARTEFAKT (engl. ARTIFACT) hat folgende Eigenschaften:

- BEZEICHNUNG (engl. DESIGNATION): Eine textuelle Bezeichnung des Artefakts
- BESCHREIBUNG (engl. DESCRIPTION): Eine textuelle Beschreibung des Artefakts
- FUNDORT (engl. LOCATION): Eine textuelle Beschreibung des Orts
- FUNDZEITPUNKT (engl. DATE OF DISCOVERY): Ein genauer Zeitpunkt, kodiert als `java.util.Date`
- TYP (engl. TYPE): Einer aus SCHMUCK, WERKZEUG, STATUETTE (engl. JEWELLERY, TOOL, FIGURINE)
- GRÖSSE (engl. SIZE): Eine Größenangabe mit LÄNGE, BREITE und HÖHE (engl. LENGTH, WIDTH and HEIGHT), sowie dem GEWICHT (engl. WEIGHT). Alle Zahlen sollen mit Nachkommastellen angegeben werden können und mit möglichst großer Präzision gespeichert werden. Eine Methode `weightPerVolume()` soll das Gewicht pro Volumeneinheit berechnen.

Es gibt außerdem ZEITALTER (engl. PERIOD). Artefakte sollen miteinander vergleichbar sein, aber nur mit anderen Artefakten des selben Zeitalters.

a) Vervollständigen Sie die unten skizzierte Klasse `Artifact` so, dass die in der Aufzählung genannten Eigenschaften repräsentiert werden können. Repräsentieren Sie Artefakt-Typ und Größe durch jeweils einen geeigneten Typ. Sie dürfen dazu keine Klassen o.ä. *außerhalb* von `Artifact` anlegen. Sie müssen keine Getter- oder Setter-Methoden angeben.

b) Ergänzen Sie die Klasse `Period` so, dass der Java-Compiler die drei Zeitalter-Klassen ohne Fehlermeldung akzeptiert.

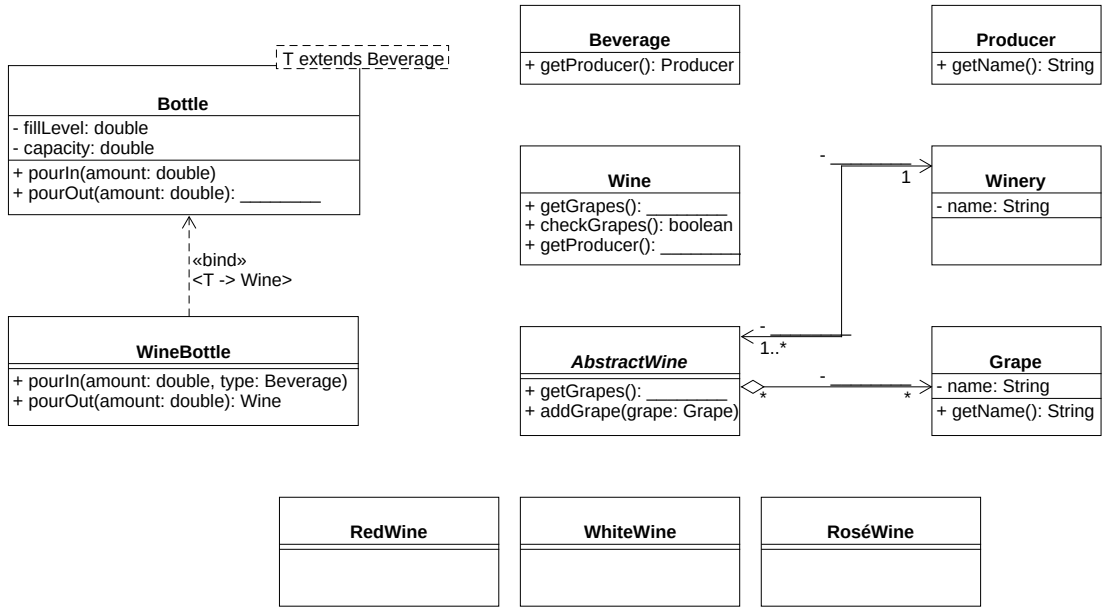
c) Erweitern Sie die Klasse `Artifact` so, dass ein Artefakt immer zu einem Zeitalter gehört, und dass nur Artefakte mit anderen Artefakten des selben Zeitalters verglichen werden können. Verwenden Sie dazu das bekannte Interface `Comparable<T>`.
Implementieren Sie die Methode `compareTo` basierend auf dem Gewicht pro Volumeneinheit der beiden Artefakte. Sie dürfen dazu die Methode `Double.compare(double d1, double d2)` verwenden.

```
1  public class Artifact
2
3
4
5    protected              designation;
6
7    protected              description;
8
9    protected              location;
10
11   protected              dateOfDiscovery;
12
13   protected              type;
14
15   protected              size;
16
17
18
19
```

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45    @Override
46    public int compareTo(
47
48
49
50
51
52  }
53
54  class abstract class Period {
55
56
57
58
59
60  }
61
62  public class AncientPeriod extends Period {
63
64    public AncientPeriod() { name = ”Ancient”; }
65
66  }
67
68  public class MedievalPeriod extends Period {
69
70    public MedievalPeriod() { name = ”Medieval”; }
71
72  }
```

Aufgabe 2: *Klassendiagramme* **(15 Punkte)**

Gegeben ist folgendes UML-Klassendiagramm:



Hier wird Wein modelliert: WEIN (engl. WINE) ist eine Spezialisierung von GETRÄNKEN (engl. BEVERAGE). Beide dürfen in dieser allgemeinen Form nicht instanziiert werden. Eine abstrakte Klasse `AbstractWine` spezialisiert den Wein. `AbstractWine` ist wiederum eine Generalisierung von ROTWEIN, WEISSWEIN und ROSÉWEIN (engl. RED WINE, WHITE WINE und ROSÉ WINE).

Getränke haben einen HERSTELLER (engl. PRODUCER), bei Weinen ist dieser Hersteller ein WEINGUT (engl. WINERY). Weine bestehen aus TRAUBEN (engl. GRAPES). Es kann zu jedem Wein geprüft werden, ob die Traubensorten zur Farbe des Weins passen (`checkGrapes`).

FLASCHEN (engl. BOTTLE) können Getränke enthalten, die man in die Flasche hinein und wieder heraus SCHÜTTEN (engl. POUR) kann. Beim Herausschütten wird der Typ des Getränks zurückgegeben.

Vervollständigen Sie das UML-Diagramm. Ergänzen Sie insbesondere fehlende Vererbungsbeziehungen, Typ-Klassifikatoren, Rückgabetypen, Rollenbezeichner und Methoden.

Aufgabe 3: *Datenstrukturen* **(1 + 2 + 11 + 7 Punkte)**

Der bekannte Kunstsammler *Robert MacDougal* will eine Datenbank aufbauen, in der er schnell für jedes beliebige MUSEUM (engl. MUSEUM) eine – selbstverständlich duplikatenfreie – Sammlung der dort ausgestellten KUNSTWERKE (engl. ARTWORK) erhalten kann.

Ein Museum wird eindeutig über seinen NAMEN (engl. NAME) und seinen ORT (engl. LOCATION) identifiziert, die beide textuell angegeben werden sollen.

Ein Kunstwerk wird eindeutig über seinen TITEL (engl. TITLE), seinen KÜNSTLER (m/w/d) (engl. ARTIST) und sein ENTSTEHUNGSDATUM (engl. CREATION DATE) identifiziert. Titel und Künstler werden textuell repräsentiert, das Datum als `java.util.Date`. Ein Kunstwerk „weiß“ außerdem, in welchem Museum es hängt.

a) Ergänzen Sie die Klasse `Museum` um die geforderten Daten. Sie müssen keine Getter- oder Setter-Methoden angeben. Sie müssen weder die `hashCode` noch die `equals` Methode überschreiben.

b) Ergänzen Sie die Klasse `Artwork` um die geforderten Daten. Sie müssen keine Getter- oder Setter-Methoden angeben. Sie müssen weder die `hashCode` noch die `equals` Methode überschreiben.

c) Ergänzen Sie die Klasse `ArtDatabase` um die nötigen Attribute zur Speicherung der Museen und der dazugehörigen Kunstwerke. Verwenden Sie neben den Klassen `Museum` und `Artwork` *ausschließlich* vom Java-JDK definierte Typen wie `Collection`, `List`, `Map` usw. Sie dürfen davon ausgehen, dass in den Klassen `Museum` und `Artwork` korrekte und sinnvolle Implementierungen der `hashCode` und der `equals` Methoden existieren.
Vervollständigen Sie die Methoden zum HINZUFÜGEN (engl. ADD) eines Kunstwerks zu einem Museum und zum ABFRAGEN (engl. QUERY) aller Kunstwerke eines Museums.

d) Schreiben Sie eine Methode `isConsistent(): boolean`, welche die Datenbank auf KONSISTENZ (engl. CONSISTENCY) prüft. Eine Datenbank ist genau dann konsistent, wenn kein Kunstwerk gleichzeitig in mehr als einem Museum ausgestellt wird.

```
1  public class Museum {
2
3
4
5
6  }
7
8  public class Artwork {
9
10
11
12
13
14
15
16
17 }
18
19 public class ArtDatabase {
20
21
22
23
24
25   public void add(Museum m, Artwork a) {
26
27
28
```

```
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47      }
48
49      public Collection<Artwork> query(Museum m) {
50
51
52
53
54
55
56      }
57
58      public boolean isConsistent() {
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80      }
81  }
```

Aufgabe 4: *Synchronisation* **(4 + 4 + 4 + 4 Punkte)**

Kreuzen Sie für jede Aussage entweder *Ja* oder *Nein* an. Korrekte Antworten werden positiv gewertet, falsche Antworten negativ, fehlende Antworten gehen nicht in die Bewertung ein. Sie können in einer Teilaufgabe *keine* negativen Punkte erreichen, auch nicht bei überwiegend falschen Antworten. Falsche Antworten führen innerhalb einer Teilaufgabe zwar zu Punktabzug, haben aber keinen Einfluss auf andere Teilaufgaben und können auch in der gleichen Teilaufgabe nicht zu einer negativen Punktezahl führen.

Hinweis: Versuchen Sie nicht zu raten. Damit verlieren Sie sonst ggf. Punkte aus richtigen Antworten.

a) Welche der folgenden Aussagen sind wahr (ja) bzw. falsch (nein)?

Ja	Nein	
[ ]	[ ]	Nebenläufigkeit ist auch ohne mehrere CPUs oder CPU-Kerne möglich.
[ ]	[ ]	Der Java-Scheduler handelt *fair*, lässt also alle Threads ähnlich lange warten.
[ ]	[ ]	Der Java-Scheduler kann einen Thread zu (fast) jedem Zeitpunkt unterbrechen.
[ ]	[ ]	Der Java-Scheduler folgt einer festen Reihenfolge beim Aktivieren von Threads.

b) Welche der folgenden Aussagen sind wahr (ja) bzw. falsch (nein)?

Ja	Nein	
[ ]	[ ]	LIFELOCK ist ein anderes Wort für DEADLOCK.
[ ]	[ ]	Synchronisation (z.B. per **`synchronized`**) macht Deadlocks unmöglich.
[ ]	[ ]	Sobald mehrere Threads laufen, muss immer synchronisiert werden.
[ ]	[ ]	Kritische Abschnitte können entstehen, sobald mindestens ein Thread schreibend auf gemeinsame Ressourcen zugreift.

c) Betrachten Sie die folgende Klasse `Sync1`:

```
1  public class Sync1 {
2
3    private static int value = 500;
4    public static void main(String[] args) {
5      new Thread(() -> {
6        while (value >= 0) {
7          value = value * 2;
8        }
9      }).start();
10     new Thread(() -> {
11       while (value >= 0) {
12         value = value / 2;
13       }
14     }).start();
15     new Thread(() -> {
16       while (value >= 0) {
17         value = value - 1;
18       }
19     }).start();
20   }
21 }
```

Welche der folgenden Aussagen sind wahr (ja) bzw. falsch (nein)?

Ja	Nein	
[ ]	[ ]	Der Code compiliert nicht.
[ ]	[ ]	Der Code stürzt mit einer `ConcurrentModificationException` ab.
[ ]	[ ]	Der Code terminiert mit hoher Wahrscheinlichkeit in endlicher Zeit.
[ ]	[ ]	Das Programm terminiert, während die Threads im Hintergrund weiterlaufen.

d) Betrachten Sie die folgende Klasse Sync2:

```
1  public class Sync2 {
2
3    private static Object lock = new Object();
4    private static int value = 500;
5    public static void main(String[] args) {
6      new Thread(() -> {
7        synchronized(lock) {
8          while (value >= 0) {
9            value = value + 1;
10         }
11       }
12     }).start();
13     new Thread(() -> {
14       synchronized(lock) {
15         while (value >= 0) {
16           value = value - 1;
17         }
18       }
19     }).start();
20   }
21 }
```

Welche der folgenden Aussagen sind wahr (ja) bzw. falsch (nein)?

Ja	**Nein**	
[ ]	[ ]	In beiden Threads wird der gesamte kritische Abschnitt synchronisiert.
[ ]	[ ]	`value` wird *im Wechsel* so lange erhöht bzw. erniedrigt, bis es kleiner als Null wird.
[ ]	[ ]	Wäre der Wertebereich von `value` unendlich, so würde das Programm nicht terminieren.
[ ]	[ ]	Der Code terminiert garantiert in endlicher Zeit.

Aufgabe 5: *Backtracking* **(15 Punkte)**

Helfen Sie der bekannten Entdeckerin *Laura Craft*, einen Weg durch ein Labyrinth zu finden!

Gegeben ist eine Klasse `Labyrinth` mit einem Eingangsraum `entrance` und einem Ausgangsraum `exit`. Diese beiden Räume sind durch eine Kette von weiteren Räumen verbunden (Attribut `adjacents` der Klasse `Room`). Um die Erstellung des Labyrinths müssen Sie sich *nicht* kümmern.

Schreiben Sie eine Methode `findPath(): boolean`, die einen Weg vom Eingangs- zum Ausgangsraum finden soll. Die Länge des Wegs ist dabei unerheblich. Die Suche kann also beim ersten gefundenen Weg abgebrochen werden, es muss *nicht* der kürzeste Weg gefunden werden. Falls ein solcher Weg existiert, gibt die Methode **`true`** zurück, sonst **`false`**.

Gleichzeitig soll die Methode den gefundenen Weg im Attribut `path` ablegen, so dass Frau Craft nach erfolgreichem Aufruf von `findPath` den in `path` gespeicherten Weg durch das Labyrinth nehmen kann.

Verwenden Sie **Backtracking**, um den Weg zu finden. Schreiben Sie dazu eine rekursive Hilfsmethode `findPath(r: Room): boolean`, welche den Weg ausgehend vom gegebenen Raum `r` weiter sucht. Diese Hilfsmethode soll von der öffentlichen Methode `findPath()` verwendet werden.

Das Attribut `path` ist vom Typ `LinkedList`, welche u.a. die Methoden `addFirst()`, `removeFirst()`, `addLast()`, `removeLast()` und `clear()` zum Hinzufügen bzw. Entfernen eines Eintrags am Anfang der Liste, zum Hinzufügen bzw. Entfernen eines Eintrags am Ende der Liste, und zum Löschen aller Listeneinträge bereitstellt.

Beachten Sie beim Finden des Wegs auch, dass Sie bei Zyklen im Labyrinth nicht ewig im Kreis laufen. Dazu können Sie bereits besuchte Räume als `visited` markieren. Die Methode `clearVisited()` setzt die `visited`-Attribute aller Räume wieder auf **`false`** zurück.

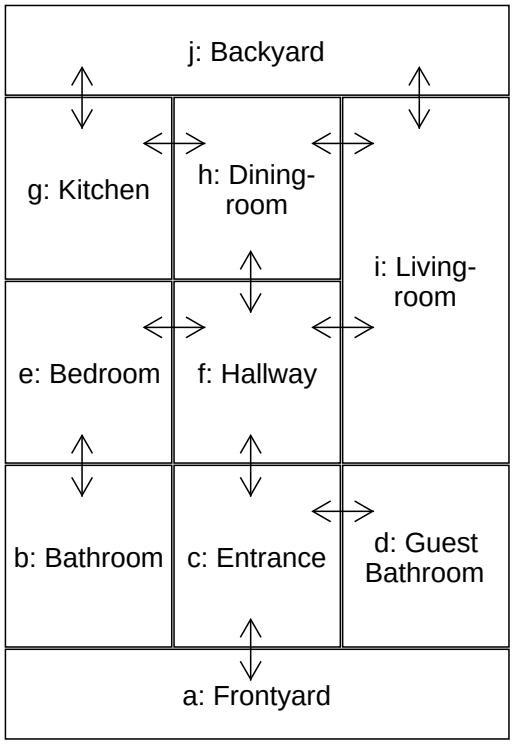
```
1  public class Labyrinth {
2
3    private Room entrance;
4    private Room exit;
5    private LinkedList<Room> path = new LinkedList<>();
6    // Getters and Setters omitted
7
8    public void init() {
9      // labyrinth is created here by cheap minotaur labour
10   }
11
12   private void clearVisited() {
13     // mark all rooms as unvisited
14   }
15
16   public boolean findPath() {
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

```
33
34
35
36
37    }
38
39    private boolean findPath(Room r) {
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75    }
76
77    public class Room {
78      private String name;
79      private List<Room> adjacents = new ArrayList<>();
80      private boolean visited;
81      // Getters and Setters
82    }
83
84    public static void main(String[] args) {
85      Labyrinth lab = new Labyrinth();
86      lab.init();
87      lab.findPath();
88    }
89  }
```

Aufgabe 6: *Prolog* **(7 + 6 Punkte)**

a) Der bekannte Künstler M. C. Escher möchte einen Plan für sein neues Haus in Prolog formalisieren. Den Grundriss sehen Sie rechts, die Verbindungen zwischen den Räumen sind durch Pfeile markiert. Der Einfachheit halber gelten auch Vorgarten und Garten (frontyard und backyard) als Räume.
Schreiben Sie eine Wissensbasis in Prolog, die alle 10 genannten Räume (`room`) enthält, sowie die 11 Verbindungen (`connection` oder `con`) zwischen je zwei Räumen. Es genügt, wenn Sie jeden Raum über den gegebenen Buchstaben benennen, Sie müssen die Namen nicht ausschreiben (z.B. `a` statt `a: Frontyard`). Schreiben Sie außerdem folgende Regeln:

- `connected(X, Y)`: Gilt, wenn X von Y aus *direkt* erreichbar ist oder umgekehrt, d.h. wenn die beiden Räume direkt verbunden sind.
- `reachable(X, Y)`: Gilt, wenn X von Y aus *direkt oder indirekt* erreichbar ist oder umgekehrt, d.h. wenn die beiden Räume direkt oder indirekt verbunden sind.



b) Schreiben Sie eine dreistellige Relation `delete`, welche ein Listenelement X, eine Liste und eine Liste ohne das Element X verknüpft. Für ein gegebenes Element und eine gegebene Liste soll die Relation – informell gesprochen – das Element aus der Liste löschen.