Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

Fach Informatik

Herbst 2020

66116

Datenbanksysteme / Softwaretechnologie (vertieft)

Aufgabenstellungen mit Lösungsvorschlägen

Aufgabenübersicht

Thema Nr. 1

Teilaufgabe Nr. 1

Aufgabe 1 [Verifikation]

(a) Definieren Sie die Begriffe *„partielle Korrektheit"* und *„totale Korrektheit"* und grenzen Sie sie voneinander ab.

partielle Korrektheit Ein Programmcode wird bezüglich einer Vorbedingung P und einer Nachbedingung Q partiell korrekt genannt, wenn bei einer Eingabe, die die Vorbedingung P erfüllt, jedes Ergebnis die Nachbedingung Q erfüllt. Dabei ist es noch möglich, dass das Programm nicht für jede Eingabe ein Ergebnis liefert, also nicht für jede Eingabe terminiert.

totale Korrektheit Ein Code wird total korrekt genannt, wenn er partiell korrekt ist und zusätzlich für jede Eingabe, die die Vorbedingung P erfüllt, terminiert. Aus der Definition folgt sofort, dass total korrekte Programme auch immer partiell korrekt sind.

(b) Geben Sie die Verifikationsregel für die abweisende Schleife `while(b) { A }` an.

Eine abweisende Schleife ist eine while-Schleife, da die Schleifenbedingung schon bei der ersten Prüfung falsch sein kann und somit die Schleife abgewiesen wird.

Um die schwächste Vorbedingung eines Ausdrucks der Form „`while(b) { A }`" zu finden, verwendet man eine *Schleifeninvariante*. Sie ist ein Prädikat für das

$$\{I \wedge b\} A \{I\}$$

gilt. Die Schleifeninvariante gilt also sowohl vor, während und nach der Schleife.

(c) Erläutern Sie kurz und prägnant die Schritte zur Verifikation einer abweisenden Schleife mit Vorbedingung P und Nachbedingung Q.

Schritt 0: Schleifeninvariante I finden

Schritt 1: I gilt vor Schleifenbeginn,
d.h: $P \Rightarrow \mathrm{wp}(\texttt{"Code vor Schleife"}, I)$

Schritt 2: I gilt nach jedem Schleifendurchlauf
d.h. $I \wedge b \Rightarrow \mathrm{wp}(\texttt{"Code in der Schleife"}, I)$

Schritt 3: Bei Terminierung der Schleife liefert Methode das gewünschte Ergebnis,
d.h. $I \wedge \neq b \Rightarrow \mathrm{wp}(\texttt{"Code nach der Schleife"}, Q)$

(d) Wie kann man die Terminierung einer Schleife beweisen?

Zum Beweis der Terminierung einer Schleife muss eine Terminierungsfunktion T angeben werden:

$$T\colon V \to \mathbb{N}$$

V ist eine Teilmenge der Ausdrücke über die Variablenwerte der Schleife

Die Terminierungsfunktion muss folgende Eigenschaften besitzen:

- Ihre Werte sind natürliche Zahlen (einschließlich 0).
- Jede Ausführung des Schleifenrumpfs verringert ihren Wert (streng monoton fallend).
- Die Schleifenbedingung ist false, wenn $T = 0$.

T ist die obere Schranke für die noch ausstehende Anzahl von Schleifendurchläufen.

Beweise für Terminierung sind nicht immer möglich!

(e) Geben Sie für das folgende Suchprogramm die nummerierten Zusicherungen an. Lassen Sie dabei jeweils die invariante Vorbedingung P des Suchprogramms weg. Schreiben Sie nicht auf dem Aufgabenblatt!

$P \equiv n > 0 \wedge a_0 \dots a_{n-1} \in \mathbb{Z}^n \wedge m \in \mathbb{Z}$

```
18      int i = -1;
19      // (1)
20      int j = 0;
21      // (2)
22      while (i == -1 && j < n) // (3)
23      { // (4)
24        if (a[j] == m) {
25          // (5)
26          i = j;
27          // (6)
28        } else {
29          // (7)
30          j = j + 1;
31          // (8)
32        }
33        // (9)
34      }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/Verfikation.java`

$Q \equiv P \wedge (i = -1 \wedge \forall 0 \leq k < n : a_k \neq m) \vee (i \geq 0 \wedge a_i = m)$

In dieser Aufgabenstellung fällt die Vorbedingung mit der Invariante zusammen. Es muss kein wp-Kalkül berechnet werden, sondern „nur“ die Zuweisungen nachverfolgt werden, um zum Schluss die Nachbedingung zu erhalten.

1. $(i = -1) \wedge P$
2. $(i = -1) \wedge (j = 0) \wedge P$
3. $(i = -1) \wedge (0 \leq j < n) \wedge P$
4. $(i = -1) \wedge (0 \leq j < n) \wedge P$
5. $(i = -1) \wedge (a_j = m) \wedge (0 \leq j < n) \wedge P$
6. $(i \geq 0) \wedge i = j \wedge (a_j = m) \wedge (0 \leq j < n) \wedge P$
7. $(i = -1) \wedge (\forall 0 \leq j < n : a_j \neq m) \wedge P$
8. $(i = -1) \wedge (\forall 0 < j \leq n : a_j \neq m) \wedge P$
9. $((i = -1) \wedge (\forall 0 \leq k < j : a_k \neq m)) \vee ((i \geq 0) \wedge (a_i = m)) \wedge P$

Code-Beispiel eingebaut in eine Java-Klasse

```
3   public class Verfikation {
4
5     /**
6      * Suche in einem Feld nach einem Wert.
7      *
8      * @param a Ein Feld in dem nach einem Werte gesucht werden soll.
9      * @param m Der gesuchte Wert.
10     *
11     * @return Falls der Wert gefunden wurde, wird die Index-Nummer im
12     * Feld ausgegeben. Wenn der Wert nicht gefunden wurde, wird -1
13     * ausgegeben.
14     */
15    public static int sucheWertInFeld(int[] a, int m) {
16      int n = a.length;
17
18      int i = -1;
19      // (1)
20      int j = 0;
```

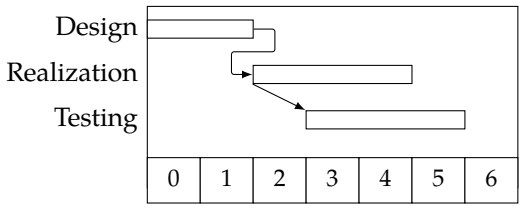
```java
    // (2)
    while (i == -1 && j < n) // (3)
    { // (4)
      if (a[j] == m) {
        // (5)
        i = j;
        // (6)
      } else {
        // (7)
        j = j + 1;
        // (8)
      }
      // (9)
    }

    return i;
  }

  public static void main(String[] args) {
    int[] a = new int[10];
    a[4] = 3;
    System.out.println(sucheWertInFeld(a, 3)); // 4
    System.out.println(sucheWertInFeld(a, 7)); // -1
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/Verfikation.java`

Aufgabe 2 [Projektplanung]

Die Planung eines Softwareprojekts kann z. B. in Form von Gantt-Diagrammen oder CPM-Netzwerken (kritischer Pfad Methode) festgehalten werden.

Folgendes Gantt-Diagramm zeigt einen Teil der Projektplanung in einem klassischen Softwareentwicklungsprozess:

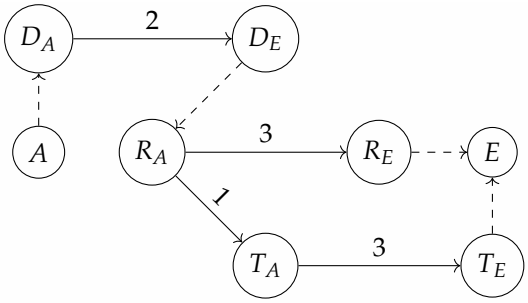


(a) Im Diagramm werden 3 Phasen aus dem klassischen Softwareentwicklungsprozess genannt. Welche Phase sollte dem Design (Entwurf) immer vorangehen?

> Die Anforderungsdefinition

(b) Wandeln Sie das Gantt-Diagramm in ein CPM-Netzwerk um. Fügen Sie dazu einen zusätzlichen Start- und Endknoten hinzu. Das Ende des Projekts ist durch das Ende aller Aktivitäten bedingt.

> D_A Design Anfang
> R_A Realization Anfang
> T_A Testing Anfang
> D_E Design Ende
> R_E Realization Ende
> T_E Testing Ende



(c) Welche im obigen Gantt-Diagramm nicht enthaltenen Beziehungsarten zwischen Aktivitäten können in einem Gantt-Diagramm noch auftreten? Nennen Sie auch deren Bedeutung.

Diese Beziehungsarten sind im obigen Gantt-Diagramm vorhanden:

Normalfolge EA: *end-to-start relationship* Anordnungsbeziehung vom Ende eines Vorgangs zum Anfang seines Nachfolgers.

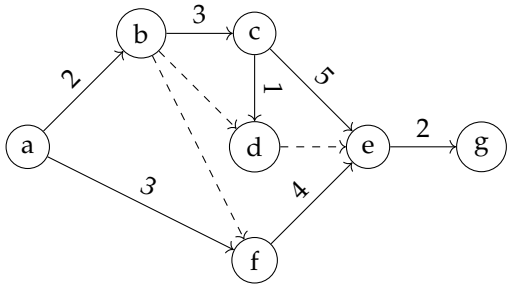
Anfangsfolge AA: *start-to-start relationship* Anordnungsbeziehung vom Anfang eines Vorgangs zum Anfang seines Nachfolgers.

Diese Beziehungsarten sind im obigen Gantt-Diagramm *nicht* vorhanden:

Endefolge EE: *finish-to-finish relationship* Anordnungsbeziehung vom Ende eines Vorgangs zum Ende seines Nachfolgers.

Sprungfolge AE: *start-to-finish relationship* Anordnungsbeziehung vom Anfang eines Vorgangs zum Ende seines Nachfolgers

Gegeben sei nun das folgende CPM-Netzwerk:



(d) Geben Sie für jedes Ereignis die früheste Zeit an.

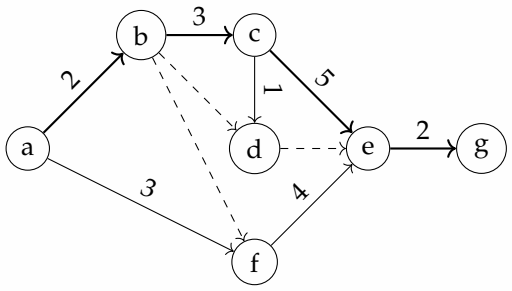
i	Nebenrechnung	FZ_i
a		0
b		2
c		5
d	$\max(2_b, 6_c)$	6
e	$\max(6_d, 10_e, 7_f)$	10
f	$\max(3_f, 2_b)$	3
g		12

(e) Geben Sie für jedes Ereignis die späteste Zeit an.

i	Nebenrechnung	SZ_i
a		0
b	$\min(2_c, 10_d, 6_f)$	2
c	$\min(9_d, 5_e)$	5
d		10
e		10
f		6
g		12

(f) Geben Sie einen kritischen Pfad durch das Netz an! Wie wirkt sich eine Verzögerung von 5 Zeiteinheiten auf dem kritischen Pfad auf das Projektende aus?

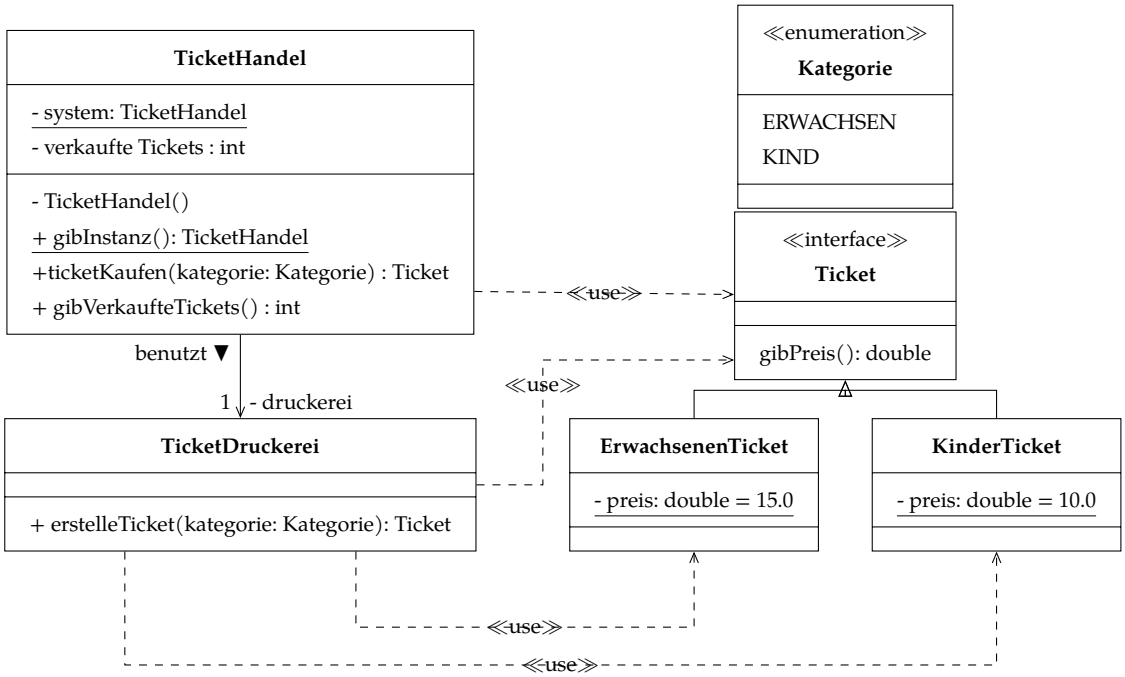
i	a	b	c	d	e	f	g
FZ_i	0	2	5	6	10	3	12
SZ_i	0	2	5	10	10	6	12
GP	0	0	0	3	0	3	0



Kritischer Pfad: a $\rightarrow$ b $\rightarrow$ c $\rightarrow$ e $\rightarrow$ g

Das Projekt verlängert sich um 5 Zeiteinheiten.

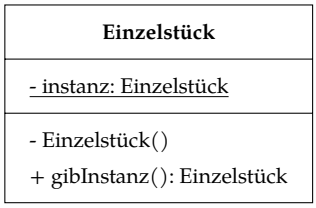
Aufgabe 3 [Ticket-Handel]



Ihnen sei ein UML-Klassendiagramm zu folgendem Szenario gegeben. Ein Benutzer (nicht im Diagramm enthalten) kann über einen `TicketHandel` Tickets erwerben. Dabei muss der Benutzer eine der zwei Ticketkategorien angeben. Das Handelsystem benutzt eine `TicketDruckerei`, um ein passendes `Ticket` für den Benutzer zu erzeugen.

(a) Im angegebenen Klassendiagramm wurden zwei unterschiedliche Entwurfsmuster verwendet. Um welche Muster handelt es sich? Geben Sie jeweils den Namen des Musters sowie die Elemente des Klassendiagramms an, mit denen diese Muster im Zusammenhang stehen. ACHTUNG: Es handelt sich dabei *nicht* um das *Interface*- oder das *Vererbungs*muster.

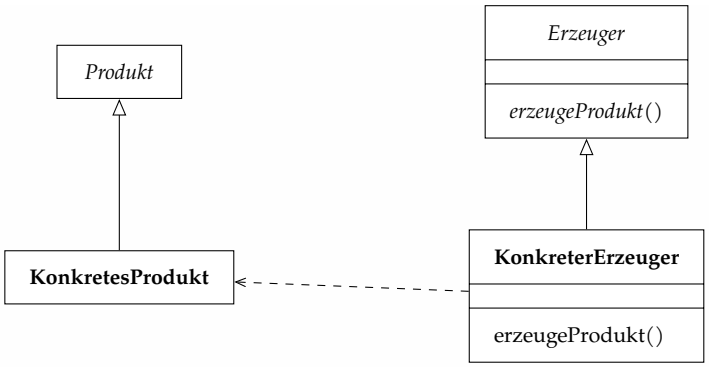
Einzelstück (Singleton)



Einzelstück (Singleton) stellt eine statische Methode bereit, mit deren Hilfe die Klienten nur auf eine einzige Instanz der Klasse zugreifen können.

Klasse	Akteur
`TicketHandel`	Einzelstück

Fabrikmethode (Factory Method)



Produkt Das Produkt ist der Basistyp (Klasse oder Schnittstelle) für das zu erzeugende Produkt.

KonkretesProdukt KonkretesProdukt implementiert die Produkt-Schnittstelle.

Erzeuger Der Erzeuger deklariert die Fabrikmethode, um ein solches Produkt zu erzeugen und kann eine Default-Implementierung beinhalten.

KonkreterErzeuger KonkreterErzeuger überschreibt die Fabrikmethode, um die ihm entsprechenden konkreten Produkte zu erzeugen (z. B. indem er den Konstruktor einer konkreten Produkt-Klasse aufruft).

Klasse	Akteur
`ErwachsenenTicket`	KonkretesProdukt
`KinderTicket`	KonkretesProdukt
`TicketDruckerei`	KonkreterErzeuger
`Ticket`	Produkt
-	Erzeuger

(b) Nennen Sie zwei generelle Vorteile von Entwurfsmustern.

- Wiederverwendung einer bewährten Lösung für eine bestimmte Problemstellungen
- Verbesserung der Kommunikation unter EntwicklerInnen

(c) Geben Sie eine Implementierung der Klasse `TicketHandel` an. Bei der Methode `ticketKaufen()` wird die Anzahl der verkauften Tickets um 1 erhöht und ein entsprechendes Ticket erstellt und zurückgegeben. Beachten Sie den Hinweis auf der nächsten Seite.

```
3  public class TicketHandel {
4    private static TicketHandel system;
5    private int verkaufteTickets;
6    private TicketDruckerei druckerei;
7
8    private TicketHandel() {
9      druckerei = new TicketDruckerei();
10     verkaufteTickets = 0;
11   }
12
13   public static TicketHandel gibInstanz() {
14     if (system == null) {
15       system = new TicketHandel();
16     }
17     return system;
18   }
19
20   public Ticket ticketKaufen(Kategorie kategorie) {
21     verkaufteTickets++;
22     return druckerei.erstelleTicket(kategorie);
```

```
23    }
24
25    public int gibVerkaufteTickets() {
26      return verkaufteTickets;
27    }
28  }
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/ticket/TicketHandel.java

(d) Geben Sie eine Implementierung der Klasse `TicketDruckerei` an.

```
3   public class TicketDruckerei {
4     public Ticket erstelleTicket(Kategorie kategorie) {
5       if (kategorie == Kategorie.ERWACHSEN) {
6         return new ErwachsenenTicket();
7       } else {
8         return new KinderTicket();
9       }
10    }
11  }
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/ticket/TicketDruckerei.java

(e) Geben Sie eine Implementierung der Klasse `KinderTicket` an.

```
3   public class KinderTicket implements Ticket {
4     private static double preis = 10.0;
5
6     public double gibPreis() {
7       return preis;
8     }
9   }
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/ticket/KinderTicket.java

Hinweis: Die Implementierungen *müssen* sowohl dem Klassendiagramm, als auch den Konzepten der verwendeten Muster entsprechen. Verwenden Sie eine objektorientierte Programmiersprache, vorzugsweie *Java*. Sie müssen sich an der nachfolgenden Testmethode und ihrer Ausgabe orientieren. Die Testmethode muss mit Ihrer Implementierung ausführbar sein und sich semantisch korrekt verhalten.

Quelltext der Testmethode:

```
4     public static void main(String[] args) {
5       TicketHandel.gibInstanz().ticketKaufen(Kategorie.ERWACHSEN);
6       TicketHandel.gibInstanz().ticketKaufen(Kategorie.KIND);
7       System.out.println("Anzahl verkaufter Tickets: " + TicketHandel.gibInstanz().gibVerkaufteTickets());
8     }
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/ticket/Test.java

Konsolenausgabe:

Anzahl verkaufter Tickets: 2

Aufgabe 4 [White-Box-Testverfahren]

Diese Aufgabe behandelt *Wortpalindrome*, also Wörter, die vorwärts und rückwärts gelesen jeweils dieselbe Zeichenkette bilden, z. B. `Otto` oder `Rentner`. Leere Wortpalindrome (also Wortpalindrome der Wortlänge 0) sind dabei nicht zulässig.

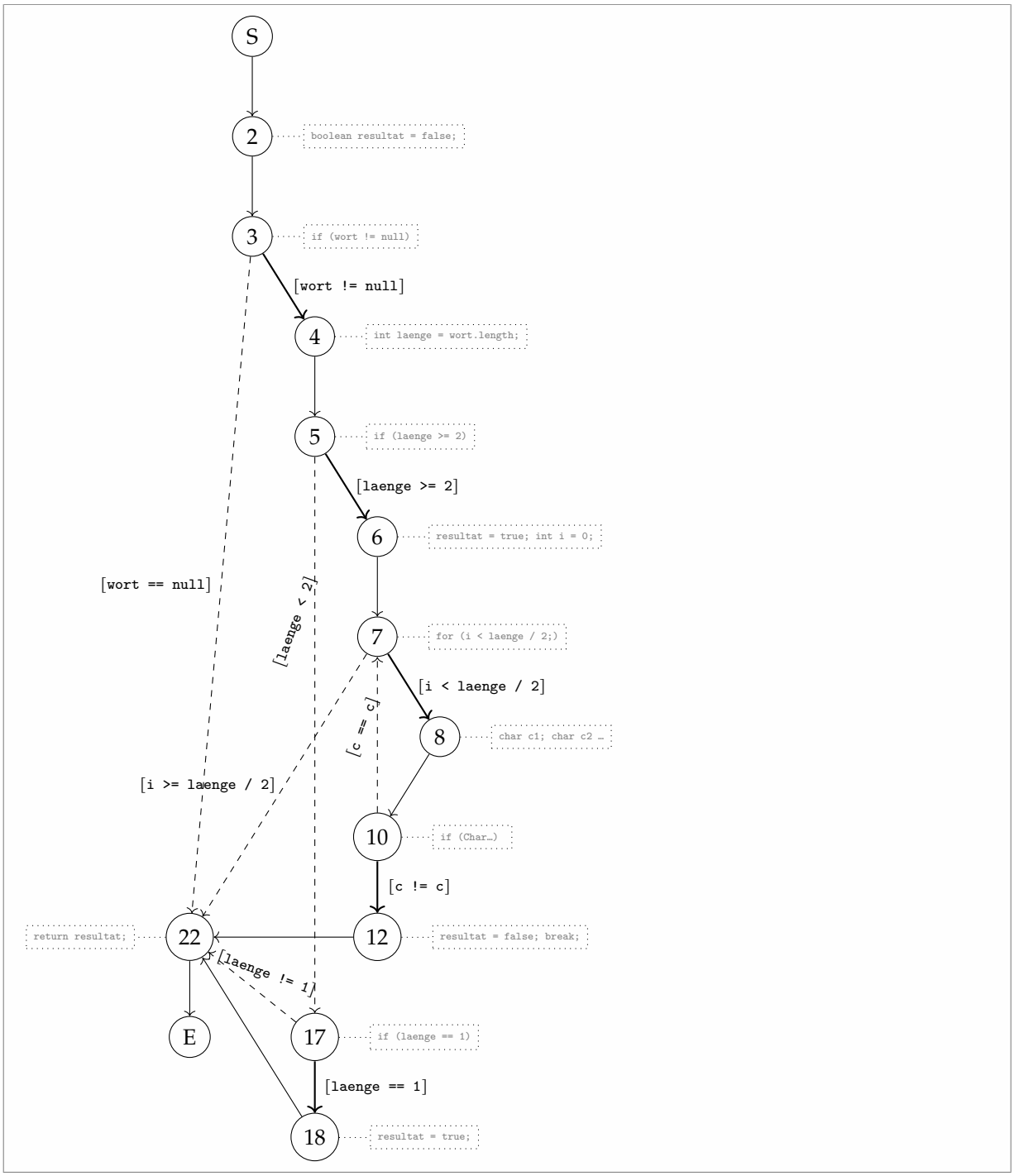
Folgende *Java-Methode* prüft, ob das übergebene Zeichen-Array ein Wortpalindrom darstellt:

```
6   public static boolean istWortpalindrom(char[] wort) { // 1
7     boolean resultat = false; // 2
8     if (wort != null) { // 3
9       int laenge = wort.length; // 4
```

```
10        if (laenge >= 2) { // 5
11          resultat = true; // 6
12          for (int i = 0; i < laenge / 2; ++i) { // 7
13            char c1 = wort[i]; // 8
14            char c2 = wort[laenge - 1 - i]; // 9
15            if (Character.toLowerCase(c1) != Character.toLowerCase(c2)) // 10
16            { // 11
17              resultat = false; // 12
18              break; // 13
19            } // 14
20          } // 15
21        } else { // 16
22          if (laenge == 1) { // 17
23            resultat = true; // 18
24          } // 19
25        } // 20
26      } // 21
27      return resultat; // 22
28    } // 23
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/Palindrom.java`

(a) Geben Sie für die Methode einen *Kontrollflussgraphen* an, wobei Sie die Knoten mit den jeweiligen Zeilennummern im Quelltext beschriften.



(b) Geben Sie eine *minimale Testmenge* an, die das Kriterium der Anweisungsüberdeckung erfüllt.

Hinweis: Eine *Testmenge* ist *minimal*, wenn es keine Testmenge mit einer kleineren Zahl von Testfällen gibt. Die Minimalität muss *nicht* bewiesen werden.

```
isWortpalindrom(new char[] { 'a' }):
(S) - (2) - (3) - (4) - (5) - (17) - (18) - (22) - (E)

isWortpalindrom(new char[] { 'a', 'b' }):
(S) - (2) - (3) - (4) - (5) - (6) - (7) - (8) - (10) - (12) - (22) - (E)
```

(c) Geben Sie eine *minimale Testmenge* an, die das Kriterium der *Boundary-Interior-Pfadüberdeckung* erfüllt.

Hinweis: Das Kriterium *Boundary-Interior-Pfadüberdeckung* beschreibt einen Spezialfall der Pfadüberdeckung, wobei nur Pfade berücksichtigt werden, bei denen jede Schleife nicht mehr als zweimal durchlaufen wird.

Es gibt noch ganz viele infeasible Pfade, die hier nicht aufgeführt werden.

Äußere Pfade

- `isWortpalindrom(null):`
  (S) - (2) - (3) - (22) - (E)
- `isWortpalindrom(new char[] { }):`
  (S) - (2) - (3) - (4) - (5) - (17) - (22) - (E)
- `isWortpalindrom(new char[] { 'a' }):`
  (S) - (2) - (3) - (4) - (5) - (17) - (18) - (22) - (E)

Grenzpfade (boundary paths, boundary test) Für Schleifen fester Lauflänge ist diese Testfallgruppe leer.

`isWortpalindrom(new char[] { 'a', 'a' }):`
(S) - (2) - (3) - (4) - (5) - (6) - (7) - (8) - (10) - (7) - (22) - (E)

`isWortpalindrom(new char[] { 'a', 'b' }):`
(S) - (2) - (3) - (4) - (5) - (6) - (7) - (8) - (10) - (12) - (22) - (E)

Innere Pfade (interior test)

- `isWortpalindrom(new char[] { 'a', 'a', 'a', 'a' }):`
  (S) - (2) - (3) - (4) - (5) - (6) - (7) - (8) - (10) - (7) - (8) - (10) - (7) - (22) - (E)
- `isWortpalindrom(new char[] { 'a', 'b', 'a', 'a' }):`
  (S) - (2) - (3) - (4) - (5) - (6) - (7) - (8) - (10) - (7) - (8) - (10) - (12) - (22) - (E)

(d) Im Falle des Kriteriums Pfadüberdeckung können minimale Testmengen sehr groß werden, da die Anzahl der Pfade sehr schnell zunimmt. Wie viele *mögliche Pfade* ergeben sich maximal für eine Schleife, die drei einseitig bedingte Anweisungen hintereinander enthält und bis zu zweimal durchlaufen wird? Geben Sie Ihren Rechenweg an (das Ergebnis alleine gibt keine Punkte).

Pro Schleifendurchlauf: $2 \cdot 2 \cdot 2 = 2^3 = 8$

Maximal 2 Schleifendurchläufe: $2 \cdot 8 = 16$

(e) Könnte für das hier abgebildete Quelltext-Beispiel auch das Verfahren der *unbegrenzten Pfadüberdeckung* (also Abdeckung aller möglicher Pfade ohne Beschränkung) als Test-Kriterium gewählt werden? Begründen Sie.

Kante 7 nach 22

Aufgabe 5 [Lebenszyklus]

(a) Nennen Sie fünf kritische Faktoren, die bei der Auswahl eines Vorgehensmodells helfen können und ordnen Sie plangetriebene und agile Prozesse entsprechend ein.

(i) Vollständigkeit der Anforderungen
- bei vollständiger Kenntnis der Anforderungen: *plangetrieben*
- bei teilweiser Kenntnis der Anforderungen: *agil*

(ii) Möglichkeit der Rücksprache mit dem Kunden
- keine Möglichkeit: *plangetrieben*
- Kunde ist partiell immer wieder involviert: *agil*

(iii) Teamgröße
- kleine Teams (max. 10 Personen): *agil*

- größere Teams: *plangetrieben*

(iv) Bisherige Arbeitsweise des Teams
- bisher feste Vorgehensmodelle: *plangetrieben*
- flexible Arbeitsweisen: *agil*

(v) Verfügbare Zeit
- kurze Zeitvorgabe: *plangetrieben*
- möglichst schnell funktionierender Prototyp verlangt: *agil*
- beide Vorgehensmodelle sind allerdings zeitlich festgelegt

Mögliche weitere Faktoren: Projektkomplexität, Dokumentation

(b) Nennen und beschreiben Sie kurz die Rollen im Scrum.

Product Owner Der Product Owner ist für die Eigenschaften und den wirtschaftlichen Erfolg des Produkts verantwortlich.

Entwickler Die Entwickler sind für die Lieferung der Produktfunktionalitäten in der vom Product Owner gewünschten Reihenfolge verantwortlich.

Scrum Master Der Scrum Master ist dafür verantwortlich, dass Scrum als Rahmenwerk gelingt. Dazu arbeitet er mit dem Entwicklungsteam zusammen, gehört aber selbst nicht dazu.

(c) Nennen und beschreiben Sie drei Scrum Artefakte. Nennen Sie die verantwortliche Rolle für jedes Artefakt.

Product Backlog Das Product Backlog ist eine geordnete Auflistung der Anforderungen an das Produkt.

Sprint Backlog Das Sprint Backlog ist der aktuelle Plan der für einen Sprint zu erledigenden Aufgaben.

Product Increment Das Inkrement ist die Summe aller Product-Backlog-Einträge, die während des aktuellen und allen vorangegangenen Sprints fertiggestellt wurden.

(d) Beschreiben Sie kurz, was ein Sprint ist. Wie lange sollte ein Sprint maximal dauern?

Ein Sprint ist ein Arbeitsabschnitt, in dem ein Inkrement einer Produktfunktionalität implementiert wird. Ein Sprint umfasst ein Zeitfenster von ein bis vier Wochen.

Thema Nr. 2

Teilaufgabe Nr. 1

Aufgabe 1

(a) Nennen Sie fünf Phasen, die im Wasserfallmodell durchlaufen werden sowie deren jeweiliges Ziel bzw. Ergebnis(-dokument).

Anforderung Lasten- und Pflichtenheft

Entwurf Softwarearchitektur in Form von Struktogrammen, UML-Diagrammen etc.

Implementierung Software

Überprüfung überarbeitete Software

Einsatz und Wartung erneut überarbeitete Software, verbesserte Wartung

(b) Nennen Sie drei Arten der Softwarewartung und geben Sie jeweils eine kurze Beschreibung an.

korrektive Wartung Korrektur von Fehlern, die schon beim Kunden in Erscheinung getreten sind

präventive Wartung Korrektur von Fehlern, die beim Kunden noch nicht in Erscheinung getreten sind

adaptive Wartung Anpassung der Software an neue Anforderungen

perfektionierende Wartung Verbesserung von Performance und Wartbarkeit und Behebung von technical depts [a]

[a] `https://de.wikipedia.org/wiki/Softwarewartung`

(c) Eine grundlegende Komponente des *Extreme Programming* ist *„Continuous Integration"*. Erklären Sie diesen Begriff und warum man davon einen Vorteil erwartet.

Die Software wird nach jeder Änderung (push) automatisch kompiliert und auch getestet. Man erwartet sich davon einen Vorteil, weil Fehler schneller erkannt werden und die aktuelle Version des Systems ständig verfügbar ist.

(d) Nennen Sie zwei Softwaremetriken und geben Sie jeweils einen Vor- und Nachteil an, der im Projektmanagement von Bedeutung ist.

Anzahl der Code-Zeilen

Vorteil: Ist sehr einfach zu bestimmen.

Nachteil: Entwickler können die Zeilenzahl manipulieren (zusätzliche Leerzeilen, Zeilen aufteilen), ohne dass sich die Qualität des Codes verändert.

Zyklomatische Komplexität

Vorteil: Trifft eine Aussage über die Qualität des Algorithmus.

Nachteil: Ist für Menschen nicht intuitiv zu erfassen. Zwei Codes, die dasselbe Problem lösen können die gleiche Zyklomatische Komplexität haben, obwohl der eine wesentlich schlechter zu verstehen ist (Spaghetticode!).

(e) Nennen und beschreiben Sie kurz drei wichtige Aktivitäten, welche innerhalb einer Sprint-Iteration von Scrum durchlaufen werden.

Sprint Planning Im Sprint Planning werden zwei Fragen beantwortet:

- Was kann im kommenden Sprint entwickelt werden?
- Wie wird die Arbeit im kommenden Sprint erledigt?

Die Sprint-Planung wird daher häufig in zwei Teile geteilt. Sie dauert in Summe maximal 2 Stunden je Sprint-Woche, beispielsweise maximal acht Stunden für einen 4-Wochen-Sprint.

Daily Scrum Zu Beginn eines jeden Arbeitstages trifft sich das Entwicklerteam zu einem max. 15-minütigen Daily Scrum. Zweck des Daily Scrum ist der Informationsaustausch.

Sprint Review Das Sprint Review steht am Ende des Sprints. Hier überprüft das Scrum-Team das Inkrement, um das Product Backlog bei Bedarf anzupassen. Das Entwicklungsteam präsentiert seine Ergebnisse und es wird überprüft, ob das zu Beginn gesteckte Ziel erreicht wurde.

(f) Erläutern Sie den Unterschied zwischen dem Product-Backlog und dem Sprint-Backlog.

Product Backlog Das Product Backlog ist eine geordnete Auflistung der Anforderungen an das Produkt.

Sprint Backlog Das Sprint Backlog ist der aktuelle Plan der für einen Sprint zu erledigenden Aufgaben.

(g) Erläutern Sie, warum eine agile Entwicklungsmethode nur für kleinere Teams (max. 10 Personen) gut geeignet ist, und zeigen Sie einen Lösungsansatz, wie auch größere Firmen agile Methoden sinnvoll einsetzen können.

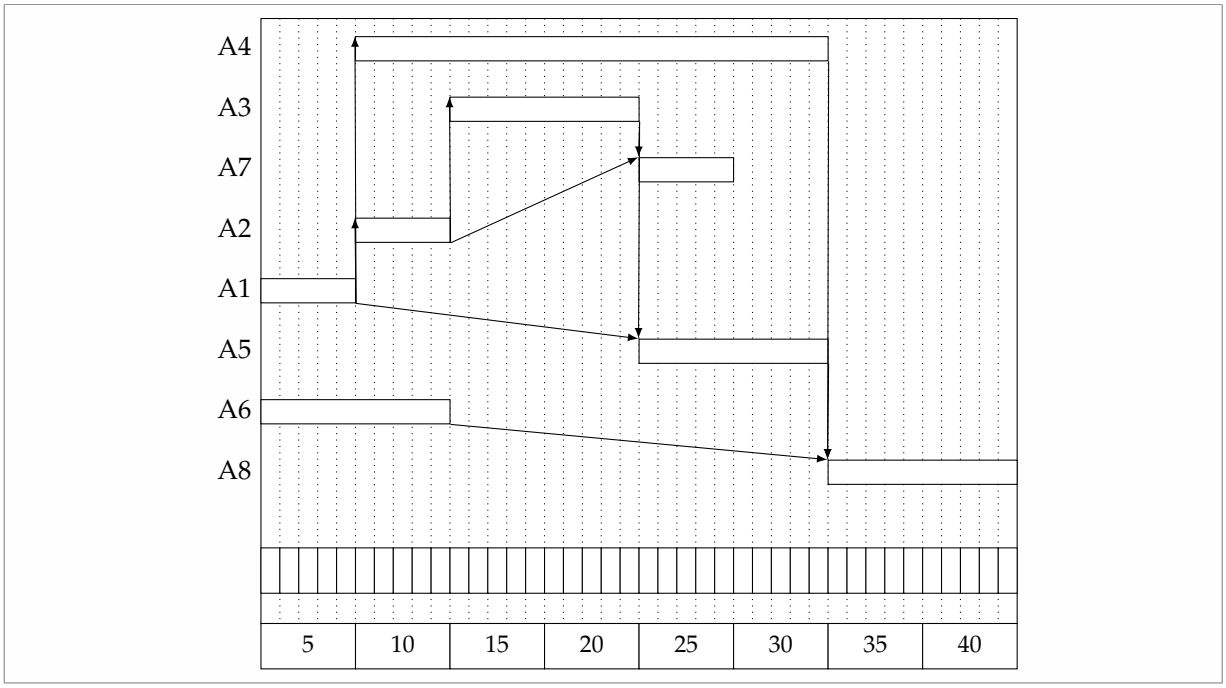
Bei agilen Entwicklungsmethoden finden daily scrums statt, die in der intendierten Kürze nur in kleinen Teams umsetzbar sind. Außerdem ist in größeren Teams oft eine Hierarchie vorhanden, die eine schnelle Entscheidungsfindung verhindert. Man könnte die große Firma in viele kleine Teams einteilen, die jeweils an kleinen (Teil-)Projekten arbeiten. Eventuell können die product owner der einzelnen Teams nach agilen Methoden zusammenarbeiten.

Aufgabe 2 [Projektmanagement]

Betrachten Sie die folgende Tabelle zum Projektmanagement:

Arbeitspaket	Dauer (Tage)	abhängig von
A1	5	
A2	5	A1
A3	10	A2
A4	25	A1
A5	10	A1, A3
A6	10	
A7	5	A2, A3
A8	10	A4, A5, A6

(a) Erstellen Sie ein Gantt-Diagramm, das die in der Tabelle angegebenen Abhängigkeiten berücksichtigt. Das Diagramm muss nicht maßstabsgetreu sein, jedoch jede Information aus der gegebenen Tabelle enthalten.



(b) Wie lange dauert das Projekt mindestens?

Es dauert mindestens 40 Tage.

(c) Geben Sie alle kritischen Pfade an.

A1 → A4 → A8
A1 → A2 → A3 → A5 → A8

(d) Bewerten Sie folgende Aussage eines Projektmanagers: „Falls unser Projekt in Verzug gerät, bringen uns neue Programmierer auch nicht weiter."

Ist der Verzug in einem Arbeitspaket, in dem genügend Pufferzeit vorhanden ist (hier A6), so hilft ein neuer Programmierer nicht unbedingt weiter. Geht allerdings die Pufferzeit zu Ende oder ist erst gar nicht vorhanden (z. B. im kritischen Pfad), so kann ein/e neue/r ProgrammerIn helfen, falls deren/dessen Einarbeitsungszeit gering ist. Muss sich der/die Neue erste komplett einarbeiten, so wird er/sie wohl auch keine große Hilfe sein.

Aufgabe 3 [Fahrkartenautomat]

Im Folgenden ist eine Systembeschreibung für einen „Fahrkartenautomat" angegeben.

Es gibt zwei Arten von Bahnfahrern: Gelegenheitsfahrer und Vielfahrer. Bei der Benutzung des Kartenautomaten kann sich grundsätzlich jeder die Hilfe anzeigen lassen und ein Beschwerdeformular ausfüllen.

Gelegenheitsfahrer haben keine Benutzer-Id und können Einzelfahrkarten auswählen und anschließend kaufen. Damit ein Kaufvorgang erfolgreich abgeschlossen wird, muss ein entsprechender Geldbetrag eingezahlt werden. Dies geschieht entweder mit Bargeld oder per EC-Karte. Nach dem Kauf eines Tickets kann man sich dafür optional eine separate Quittung drucken lassen.

Vielfahrer haben eine eindeutige Benutzer-Id mit Passwort, um sich am Automaten zu authentifizieren. Ein Vielfahrer kann sowohl eine Einzelfahrkarte als auch eine personalisierte Monatskarte erwerben. Sofern er eine Monatskarte besitzt (Information im System hinterlegt), kann er sich kostenfrei eine Ersatzfahrkarte ausstellen lassen, falls er seine Monatskarte verloren hat. Wenn die Authentifizierung oder ein Kaufvorgang fehlschlägt, soll eine entsprechende Fehlermeldung erscheinen.

(a) Geben Sie die im Text erwähnten Akteure für das beschriebene System an.

Bahnfahrer

- Gelegenheitsfahrer
- Vielfahrer

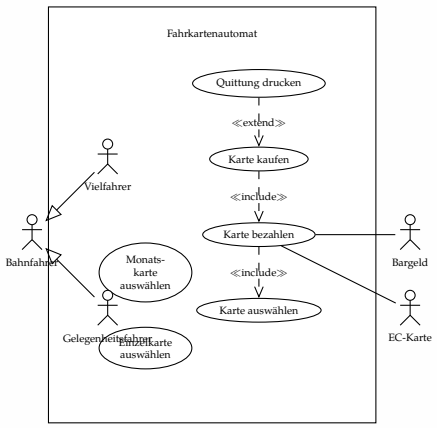
(b) Identifizieren Sie zwei weitere Stakeholder und nennen Sie dazu je zwei unterschiedliche Anwendungsfälle des Systems, in die diese involviert sind.

EC-Karte Bargeld

(c) Geben Sie mindestens sechs verschiedene Anwendungsfälle für das beschriebene System an.

- Hilfe anzeigen
- Beschwerdeformular ausfüllen
- Auswahl
  - Einzelfahrkarten
  - Monatskarte
- Einzelfahrkarten kaufen
- Zahlen
  - Bargeld
  - EC-Karte
- Quittung drucken
- Authentifizieren
  - Bargeld
  - EC-Karte

(d) Erstellen Sie aus Ihren vorherigen Antworten ein Use-Case-Diagramm für das beschriebene System, in dem die Akteure und Anwendungsfälle inkl. möglicher Generalisierungen und Beziehungen eingetragen sind. Achten Sie insbesondere auf mögliche «include»- und «extends»-Beziehungen und Bedingungen für Anwendungsfälle.



Aufgabe 4 [Objektorientierte Analyse]

Betrachten Sie das folgende Szenario:

Entwickeln Sie für einen Kunden eine einheitliche Online-Plattform, in welcher mehrere Restaurants angebunden sind. Die Nutzer des Systems sollen Essen wie Pizzen, Burger oder Pasta bestellen und dabei aus einer Liste von verschiedenen Gerichten auswählen können. Die Plattform soll zusätzliche Optionen (z. B. Lieferung

durch einen Lieferdienst oder direkte Abholung, inkl. Salat oder einer Flasche Wein) ermöglichen. Die Bestellung soll dann von der Plattform an den jeweiligen Gaststättenbetreiber gesendet werden. Die Besteller sollen zudem eine Bestätigung als Nachricht erhalten. Die jeweiligen Gerichte und Optionen haben unterschiedliche Preise, die dem Internetnutzer angezeigt werden müssen, bevor er diese auswählt. Der Endpreis muss vor der endgültigen Zahlung des Auftrags angezeigt werden. Kunden können (optional) einen Benutzeraccount anlegen und erhalten bei häufigen Bestellungen einen Rabatt.

(a) Beschreiben Sie kurz ein Verfahren, wie Sie aus der Szenariobeschreibung mögliche Kandidaten für Klassen erhalten können.

> Verfahren nach Abbott; [a]
>
> Objektorientierte Analyse und Design (OOAD)
>
> [a] `http://info.johpie.de/stufe_q1/info_01_verfahren_abbott.pdf`

(b) Beschreiben Sie kurz ein Verfahren, um Vererbungshierarchien zu identifizieren.

> Eine Begriffshierachie mit den Substantiven des Textes bilden.

(c) Geben Sie fünf geeignete Klassen für das obige Szenario an. Nennen Sie dabei keine Klassen, welche durch Basisdatentypen wie Integer oder String abgedeckt werden können.

> - Benutzer (Kunde, Gaststättenbetreiber)
> - Restaurant
> - Gericht (Pizza, Burger, Pasta)
> - ZusatzOption (Lieferung, Salat, Wein)
> - Bestellung

(d) Nennen Sie drei Klassen für das obige Szenario, die direkt durch Basisdatentypen wie Integer oder String abgedeckt werden können.

> - Nachricht
> - Preis
> - Rabatt

(e) Erstellen Sie ein Sequenzdiagramm für das gegebene System mit folgendem Anwendungsfall: Ein Nutzer bestellt zwei Pizzen und eine Flasche Wein. Beginnen Sie mit der „Auswahl des Gerichts“ bis hin zur „Bestätigung der Bestellung“ sowie „Lieferung an die Haustür“. Das Diagramm soll mindestens je einen Akteur für Benutzer (Browser), Applikation (Webserver) und Restaurant (Koch und Lieferdienst) vorsehen. Die Bezahlung selbst muss nicht modelliert werden.

Aufgabe 5 [Terme über die Rechenarten]

Wir betrachten Terme über die Rechenarten $op \in \{+, -, \cdot, \div\}$, die rekursiv definiert sind:

- Jedes Literal ist ein Term, z. B. „4“.
- Jedes Symbol ist ein Term, z. B. „x“.
- Ist t ein Term, so ist „(t)“ ein (geklammerter) Term.
- Sind t_1, t_2 Terme, so ist „$t_1\ op\ t_2$“ ebenso ein Term.

Beispiele für gültige Terme sind also „$4 + 8$“, „$4 \cdot x$“ oder „$4 + (8 \cdot x)$“.

(a) Welches Design-Pattern eignet sich hier am besten zur Modellierung dieses Sachverhalts?

Kompositum

(b) Nennen Sie drei wesentliche Vorteile von Design-Pattern im Allgemeinen.

(c) Modellieren Sie eine Klassenstruktur in UML, die diese rekursive Struktur von *Termen* abbildet. Sehen Sie mindestens einzelne Klassen für die *Addition* und *Multiplikation* vor, sowie weitere Klassen für *geklammerte Terme* und *Literale*, welche ganze Zahlen repräsentieren. Gehen Sie bei der Modellierung der Klassenstruktur davon aus, dass eine objektorientierte Programmiersprache wie Java zu benutzen ist.

(d) Erstellen Sie ein Objektdiagramm, welches den Term $t := 4 + (3 \cdot 2) + (12 \cdot y / (8 \cdot x))$ entsprechend Ihres Klassendiagramms repräsentiert.

(e) Überprüfen Sie, ob das Objektdiagramm für den in Teilaufgabe d) gegebenen Term eindeutig definiert ist. Begründen Sie Ihre Entscheidung.

(f) Die gegebene Klassenstruktur soll mindestens folgende Operationen unterstützen:

- das Auswerten von Termen,
- das Ausgeben in einer leserlichen Form,
- das Auflisten aller verwendeten Symbole. Welches Design-Pattern ist hierfür am besten geeignet?

(g) Erweitern Sie Ihre Klassenstruktur um die entsprechenden Methoden, Klassen und Assoziationen, um die in Teilaufgabe f) genannten zusätzlichen Operationen gemäß dem von Ihnen genannten Design Pattern zu unterstützen.

```java
public class Addition extends Rechenart {
  public Addition(Term a, Term b) {
    super(a, b, "+");
  }

  public double auswerten() {
    return a.auswerten() + b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Addition.java`

```java
public class Divison extends Rechenart {
  public Divison(Term a, Term b ) {
    super(a, b, "/");
  }

  public double auswerten() {
    return a.auswerten() / b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Divison.java`

```java
public class GeklammerterTerm extends Term {
  Term term;

  public GeklammerterTerm(Term term) {
    this.term = term;
  }

  public double auswerten() {
    return term.auswerten();
  }

  public void ausgeben() {
    System.out.print("(");
    term.ausgeben();
    System.out.print(")");
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/GeklammerterTerm.java`

```java
public class Literal extends Term {
  int wert;

  public Literal(int wert) {
    this.wert = wert;
  }

  public double auswerten() {
    return wert;
  }

  public void ausgeben() {
    System.out.print(wert);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Literal.java`

```java
public class Multiplikation extends Rechenart {
  public Multiplikation(Term a, Term b) {
    super(a, b, "*");
  }

  public double auswerten() {
    return a.auswerten() * b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Multiplikation.java`

```java
abstract class Rechenart extends Term {
  Term a;
  Term b;
  String operatorZeichen;

  public Rechenart (Term a, Term b, String operatorZeichen) {
    this.a = a;
    this.b = b;
    this.operatorZeichen = operatorZeichen;
  }

  public void ausgeben () {
    a.ausgeben();
    System.out.print(" " + operatorZeichen + " ");
    b.ausgeben();
  }

  abstract public double auswerten();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Rechenart.java`

```java
public class Subtraktion extends Rechenart {
  public Subtraktion(Term a, Term b) {
    super(a, b, "-");
  }

  public double auswerten() {
    return a.auswerten() - b.auswerten();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Subtraktion.java`

```java
public class Symbol extends Term {
  String name;
  public Symbol(String name) {
    this.name = name;
```

```
7     }
8
9     public double auswerten() {
10      return Klient.symbole.get(name);
11    }
12
13    public void ausgeben() {
14      System.out.print(name);
15    }
16  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Symbol.java`

```
3   abstract class Term {
4     abstract public double auswerten();
5
6     abstract public void ausgeben();
7   }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2020/herbst/rechenarten/Term.java`

Aufgabe 6

Gegeben sei folgendes Java-Programm, das mit der Absicht geschrieben wurde, den größten gemeinsamen Teiler zweier Zahlen zu berechnen:

```
1   /** Return the Greatest Common Divisor of two integer values. */
2
3   public int gcd(int a, int b) {
4     if (a < 0 || b < 0) {
5     return gcd(-b, a);
6
7     }
8     while (a != b) {
9       if (a>b){
10        a = a - b;
11      } else {
12        b = b % a;
13      }
14    }
15    return 3;
16  }
```

(a) Bestimmen Sie eine möglichst kleine Menge an Testfällen für das gegebene Programm, um (dennoch) vollständige Zweigüberdeckung zu haben.

(b) Welche Testfälle sind notwendig, um die Testfälle der Zweigüberdeckung so zu erweitern, dass eine 100% Anweisungsüberdeckung erfüllt ist? Begründen Sie Ihre Entscheidung.

(c) Beschreiben Sie zwei allgemeine Nachteile der Anweisungsüberdeckung.

(d) Das gegebene Programm enthält (mindestens) zwei Fehler. Bestimmen Sie jeweils eine Eingabe, die fehlerhaftes Verhalten des Programms verursacht. Nennen Sie den Fault und den Failure, der bei der gewählten Eingabe vorliegt. Sie müssen das Programm (noch) nicht verbessern.

(e) Geben Sie für die gefundenen zwei Fehler jeweils eine mögliche Verbesserung an. Es reicht eine textuelle Beschreibung, Code ist nicht notwendig.

(f) Erläutern Sie, warum es im Allgemeinen hilfreich sein kann, bei der Fehlerbehebung in einem größeren Programm die Versionsgeschichte miteinzubeziehen.

Teilaufgabe Nr. 2

Aufgabe 2 [Restaurant]

Gegeben ist der folgende Ausschnitt eines Schemas für die Verwaltung eines Restaurants.
Hinweis: Unterstrichene Attribute sind Primärschlüsselattribute, kursiv geschriebene Attribute sind Fremdschlässelattribute.
Restaurant : | RestaurantID : INTEGER, RestaurantName : VARCHAR (255), StadtName : VARCHAR(255), PLZ: INTEGER, 1
Küche: |
RestaurontID : INTEGER, Art : VARCHAR(255), KochPersonID : INTEGER
Straße : VARCHAR (255), Hausnummer: INTEGER, Kategorie : VARCHAR (255)
1
Stadt : [Person : |
StadtName : VARCHAR(255), PersonID : INTEGER,
Land : VARCHAR(255) Name : VARCHAR(255), 1 Vorname : VARCHAR (255), StadtName : VARCHAR(255), PLZ: INTEGER, Straße : VARCHAR(255),
Hausnummer: INTEGER
bevorzugt : | PersonID : INTEGER, Art : VARCHAR(255)
I
Die Tabelle Restaurant beschreibt Restaurants eindeutig durch ihre ID. Zudem wird der Name, die Adresse des Restaurants und die (Sterne-)Kategorie gespeichert. Küche enthält u. a. Informationen zu der Art der Küche. Dabei kann ein Restaurant mehrere Arten anbieten, z. B. italienisch, deutsch, etc. In der Tabelle Stadt werden der Name der Stadt sowie das Land verwaltet, in dem die Stadt liegt. Wir gehen davon aus, dass eine Stadt eindeutig durch ihren Namen gekennzeichnet ist. Person beschreibt Personen mit Name, Vorname und Adresse. Personen werden eindeutig durch eine ID identifiziert. Die Tabelle bevorzugt gibt an, welche Person welche Art der Küche präferiert.

Bearbeiten Sie die folgenden Teilaufgaben:

(a) Erläutern Sie kurz, warum das Attribut Art in Küche Teil des Primärschlüssels ist.

> Es kann mehr als eine Küche pro Restaurant geben.

(b) Schreiben Sie eine SQL-Anweisung, welche alle Städte findet, in denen man "deutsch" (Art der Küche) essen kann.

```sql
SELECT DISTINCT s.StadtName
FROM Stadt s, Restaurant r, Küche k
WHERE s.Stadtname = r.StadtName AND r.RestaurantID = k.RestaurantID AND Art = 'deutsch';
```

(c) Schreiben Sie eine SQL-Abfrage, die alle Personen (Name und Vorname) liefert, die kein deutsches Essen bevorzugen. Verwenden Sie keinen Join.

```sql
SELECT Name, Vorname
FROM Person
WHERE PersonID IN (SELECT DISTINCT PersonID
FROM bevorzugt
EXCEPT
SELECT DISTINCT PersonID
FROM bevorzugt
WHERE Art = 'deutsch');
```

(d) Schreiben Sie eine SQL-Abfrage, die für jede Stadt (StadtName) und Person (PersonID) die Anzahl der Restaurants ermittelt, in denen diese Person bevorzugt essen gehen würde. Es sollen nur Städte ausgegeben werden, in denen es mindestens drei solche Restaurants gibt.

```sql
SELECT r.StadtName, b.PersonID, count(DISTINCT r.RestaurantID) as Anzahl
FROM Restaurant r, bevorzugt b, Kueche k
WHERE r.RestaurantID = k.RestaurantID AND k.Art = b.Art
GROUP BY r.StadtName, b.PersonID
```

```
5  HAVING count(r.RestaurantID) >= 3;
```

(e) Schreiben Sie eine SQL-Abfrage, die die Namen aller Restaurants liefert, in denen sich die Personen mit den IDs 1 und 2 gemeinsam zum Essen verabreden können, und beide etwas zum Essen finden, das sie bevorzugen. Es sollen keine Duplikate ausgegeben werden.

```
1  CREATE VIEW Person1 AS
2  SELECT DISTINCT r.RestaurantID, r.RestaurantName
3  FROM Person p, bevorzugt b, Restaurant r, Küche k
4  WHERE r.StadtName = p.StadtName AND p.PersonID = b.PersonID
5  AND r.RestaurantID = k.RestaurantID AND k.Art = b.Art
6  AND p.PersonID = 1;
7  CREATE VIEW Person2 AS
8  SELECT DISTINCT r.RestaurantID, r.RestaurantName
9  FROM Person p, bevorzugt b, Restaurant r, Küche k
10 WHERE r.StadtName = p.StadtName AND p.PersonID = b.PersonID
11 AND r.RestaurantID = k.RestaurantID AND k.Art = b.Art
12 AND p.PersonID = 2;
13 SELECT * FROM Person1
14 INTERSECT
15 SELECT * FROM Person2;
```

Aufgabe [Relationale Algebra und Optimierung]

```
1  CREATE TABLE V (
2    Name VARCHAR(1),
3    Jahr integer
4  );
5
6  CREATE TABLE S (
7    Jahr integer
8  );
9
10 INSERT INTO V VALUES
11   ('A', 2019),
12   ('A', 2020),
13   ('B', 2018),
14   ('B', 2019),
15   ('B', 2020),
16   ('C', 2017),
17   ('C', 2018),
18   ('C', 2020);
19
20 INSERT INTO S VALUES
21   (2018),
22   (2019),
23   (2020);
```

(a) Betrachten Sie die Relation V. Sie enthält eine Spalte *Name* sowie ein dazugehörendes Jahr.

Name	Jahr
A	2019
A	2020
B	2018
B	2019
B	2020
C	2017
C	2018
C	2020

(i) Gesucht ist eine Relation S, die das folgende Ergebnis von $V \div S$ berechnet ($\div$ ist die Division der relationalen Algebra):
$V \div S$

Name
B

Welche der nachstehenden Ausprägungen für die Relation liefert das gewünschte Ergebnis? Geben Sie eine Begründung an.

i.

Jahr
2017
2018
2019
2020

ii.

Jahr
2018
2019
2020

iii.

Jahr
2017
2019
2020

iv. ii.,

iv) also weder i., noch ii., noch iii.

i.

Name

ii.

Name
C

iii.

Name
B
C

(ii) Formulieren Sie die Divisions-Query aus Teilaufgabe i. in SQL.

```
SELECT DISTINCT v1.Name FROM V as v1
WHERE NOT EXISTS (
  (SELECT s.Jahr FROM S as s)
  EXCEPT
  (SELECT v2.Jahr FROM V as v2 WHERE v2.Name = v1.Name)
);
```

1

(b) Gegeben sind die Tabellen R(A, B) und S(C, D) sowie die folgende View:

1 CREATE VIEW mv (A,C,D) AS

[1] https://www.geeksforgeeks.org/sql-division/

, SELECT DISTINCTA,C,D

» FROMR,S

« WHEREB=DANDA <> 10;

Auf dieser View wird die folgende Query ausgeführt:

, SELECT DISTINCT A , FROM mv ;» WHEREC>D:

Konvertieren Sie die Query und die zugrundeliegenden View in einen Ausdruck der relationalen Algebra in Form eines Operatorbaums. Führen Sie anschließend eine relationale Optimierung durch. Beschreiben und begründen Sie dabei kurz jeden durchgeführten Schritt.

(c) Gegeben sind die Relationen R, S und U sowie deren Kardinalitäten Tr, Ts und Tr:

R (al, a2, a3) Tr = 200 S (al, a2, a3) Ts = 100 U (ul, u2) Iv = 50

Bei der Ausführung des folgenden Query-Plans wurden die Kardinalitäten der Zwischenergebnisse mitgezählt und an den Kanten notiert.

Leiten Sie aus den Angaben im Ausführungsplan den Anteil der qualifizierten Tupel aller Prädikate her und geben Sie diese an.

Tx s0| N Ral > Vu

N R.a3 = S.a3 U N OR.al > 100 OS.al < 10

R 5

Aufgabe 4 [Entwurfstheorie]

Gegeben ist das folgende Relationenschema R in erster Normalform.
R:[A,B, C, D, E, F]
Für R gelte folgende Menge FD funktionaler Abhängigkeiten:

$$\mathrm{FA} = \Big\{ \begin{aligned} \{AC\} &\to \{DE\}, \\ \{ACE\} &\to \{B\}, \\ \{E\} &\to \{B\}, \\ \{D\} &\to \{F\}, \\ \{AC\} &\to \{F\}, \\ \{AD\} &\to \{F\}, \end{aligned} \Big\}$$

(a) R mit FD hat genau einen Kandidatenschlüssel X. Bestimmen Sie diesen und begründen Sie Ihre Antwort.

> AC ist der Kandidatenschlüssel. AC kommt in keiner rechten Seite der Funktionalen Abhängigkeiten vor.

(b) Berechnen Sie Schritt für Schritt die Hülle X^+ von $X := \{K\}$.

> (i) $AC \cup DE$
> (ii) $ACDE \cup B$ (ACE -> B)
> (iii) $ACDEB$ (E -> B)
> (iv) $ACDEB \cup F$ (D -> F)
> (v) $ACDEBF$ (AC -> F)
> (vi) $ACDEBF$ (AD -> F)

(c) Nennen Sie alle primen und nicht-primen Attribute.

prim: AC
nicht-prim: BDEF

(d) Geben Sie die höchste Normalform an, in der sich die Relation befindet. Begründen Sie.

2NF
D –> F hängt transitiv von AC ab: AC -> D, D-> F

(e) Gegeben ist die folgende Zerlegung von R:
R1 (A, C, D, E) R2 (B, E) R3 (D, F)
Weisen Sie nach, dass es sich um eine verlustfreie Zerlegung handelt.