Lehrstuhl für Software Engineering
RWTH Aachen University
Dr. Judith Michael
Steffen Hillemacher, M. Sc.
Oliver Kautz, M. Sc.

Softwaretechnik
Klausur
WS 2020/21
06.04.2021

1. Klausur
Softwaretechnik

Hinweise

Jedes Blatt der Abgabe ist in der Kopfzeile mit Ihrem Namen und Ihrer Matrikelnummer zu versehen. Lesen Sie die Aufgabenstellung einer Teilaufgabe zuerst vollständig durch, bevor sie mit der Lösung beginnen.

Außer dokumentenechten Schreibgeräten (also kein Bleistift o.ä.; außerdem kein Rot oder Grün) sind keine weiteren Hilfsmittel erlaubt.

Die Klausur besteht aus insgesamt 6 Aufgaben. Die Klausur ist bestanden, wenn 60 von 120 Punkten erreicht wurden. Die Bearbeitungszeit beträgt 120 Minuten.

Mit Ihrer Unterschrift bestätigen Sie, dass Sie sich prüfungstauglich fühlen.

Persönliche Angaben

Name, Vorname .....

Matrikelnummer ..... Studiengang .....

Unterschrift .....

Punkte

	A1 25	A2 20	A3 15	A4 28	A5 22	A6 10	Summe 120
Punkte							
Erstkorrektor							
Zweitkorrektor							

Name: Matrikelnummer:

Aufgabe 1 – Klassendiagramme (20 + 5 = 25 Punkte)

Sie befinden sich im Entwurf eines Verwaltungssystems für Feuerwehren einzelner Städte. Dabei machen Sie sich genauere Gedanken über die Datenstruktur des Systems. Die Spezifikation des Systems lautet wie folgt:

Eine Feuerwehr hat einen Namen und eine Kennung zur eindeutigen Referenzierbarkeit. Feuerwehren bestehen aus mindestens einer und bis zur vier Direktionen. Dabei ist es so, dass wenn eine Feuerwehr aufgelöst wird, auch ihre Direktionen aufgelöst werden. Direktionen sind einem Bereich zugeteilt, der entweder Nord, Ost, Süd oder West ist. Zu einer Direktion gehören Wachen. Eine Wache ist genau einer Direktion zugeteilt. Außerdem hat jede Wache einen Namen und eine Kennung. Wachen werden dabei unterteilt in Berufswachen, freiwillige Wachen und Rettungswachen. Zudem werden Berufswachen aufgeteilt in Vollzugwachen und Noteinsatzwachen (NEF). Manche der Wacharten haben einen Ausrückebereich. Berufswachen haben dabei immer genau einen Ausrückebereich, wohingegen freiwillige Wachen einen Ausrückebereich haben können. Rettungswachen haben keinen Ausrückebereich, sondern fahren überall Einsätze. Für einen Ausrückebereich wird angegeben, ob sich Hochhäuser oder Autobahnen in ihm befinden. Zusätzlich gilt, dass einem Ausrückebereich genau eine Berufswache und bis zu einer freiwilligen Wache zugeordnet sind.

Des Weiteren gehören Mitarbeiter zu einer Feuerwehr. Mitarbeiter haben einen Namen und eine Personalnummer. Es gilt, dass ein Mitarbeiter nur genau für eine Feuerwehr arbeiten kann und genau einer Wache zugeteilt ist. Wachen haben dabei mindestens einen Mitarbeiter. Jeder Mitarbeiter hat mindestens eine Stellenbezeichnung und genau eine Amtsbezeichnung. Stellenbezeichnungen besitzen ein Kürzel und für die Amtsbezeichnung eines Mitarbeiters werden die Dienstjahre geführt. Zusätzlich werden Stellenbezeichnungen unterschieden in Wachleiter, Brandmeister und Sanitäter. Manche Stellenbezeichnung schließen einander aus, so können Mitarbeiter zum Beispiel nicht Wachleiter und Brandmeister gleichzeitig sein. Für die Stellenbezeichnung Wachleiter gilt, dass ein Wachleiter genau eine Wache leitet und Wachen von genau einem Wachleiter geleitet werden. Brandmeister sind genau einer Wache zugeordnet und eine Wache hat mindestens sechs und maximal zwölf Brandmeister.

Teilaufgabe a) (20 Punkte)

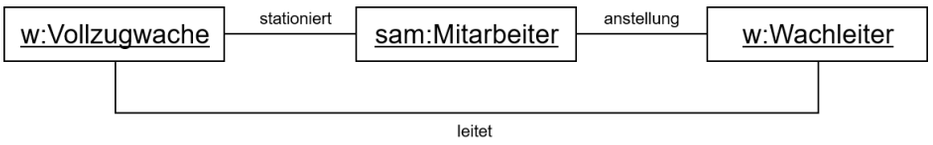
Formalisieren Sie die folgende textuelle Beschreibung des Systems, indem Sie ein geeignetes Klassendiagramm erstellen. Achten Sie darauf, dass nicht jedes Detail der Beschreibung in dem Klassendiagramm modelliert werden kann.

Name: Matrikelnummer:

Teilaufgabe b) (5 Punkte)

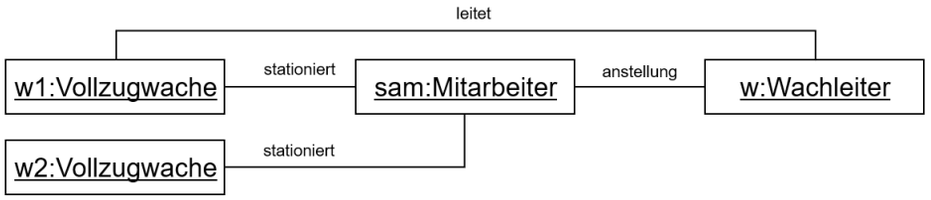
Im Folgenden sind vier Objektdiagramme geben. Geben Sie für jedes Objektdiagramm an, ob es der Spezifikation des Verwaltungssystems entspricht. Begründen Sie kurz Ihre Antwort, falls dies für ein Objektdiagramm nicht der Fall ist.

OD Diagram1



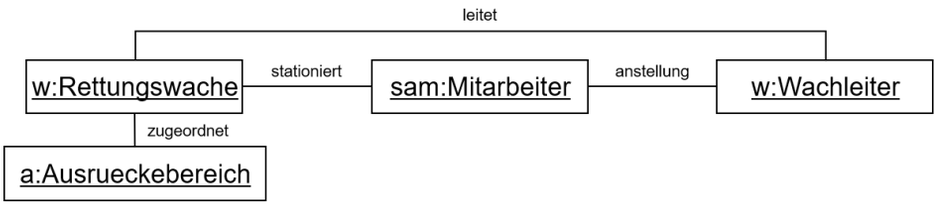
Antwort:

OD Diagram2



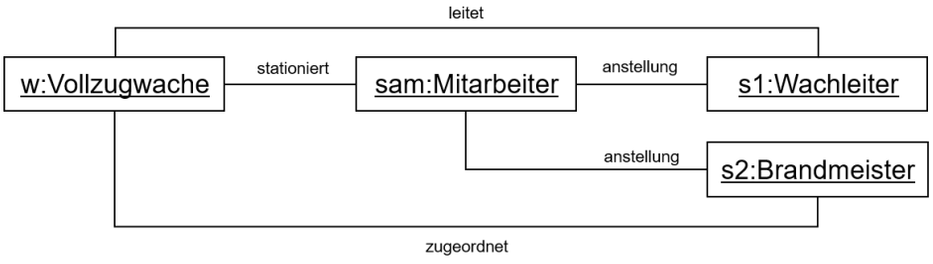
Antwort:

OD Diagram3



Antwort:

OD Diagram4



Antwort:

Name: Matrikelnummer:

Aufgabe 2 – Sequenzdiagramme (20 Punkte)

Ihre Aufgabe ist es den Ablauf der Abhandlung eines Notrufs innerhalb einer Feuerwehr zu modellieren. Erstellen Sie zu dem im Folgenden beschriebenen Ablauf ein passendes Sequenzdiagramm:

Eine Person meldet per Mobiltelefon bei der Leitstelle der Feuerwehr einen Notruf. Die Leitstelle erfragt daraufhin alle für sie relevanten Informationen von der Person. Die Person beantwortet zügig die Fragen und gibt so Informationen an die Leitstelle. Nachdem die Leitstelle alle Informationen gesammelt hat, ist das Gespräch mit der Person beendet und sie meldet auf der für den Alarm zuständigen Wache einen Alarm. Dabei übergibt die Leitstelle die zuvor gesammelten Informationen an die Wache. Als nächstes schickt die Wache ein Löschfahrzeug, das in der Wache steht, zu der Alarmierung. Dazu werden die Informationen der Leitstelle an das Fahrzeug weitergegeben. Direkt danach meldet die Wache außerdem eine Blitzmeldung an ein Löschfahrzeug, das sich noch auf dem Rückweg zur Wache befindet. Auch hier reicht die Wache die Informationen der Leitstelle an das Fahrzeug weiter. Beide Fahrzeuge schicken nach einer kurzen Zeit eine Bestätigung an die Wache.

Nach einer gewissen Zeit kommt das Fahrzeug, das die Blitzalarmierung erhalten hat, an der Einsatzstelle an und gibt eine Statusmeldung an die Wache zurück. Diese bestätigt die Meldung und bestellt danach das andere Löschfahrzeug ab. Dieses Fahrzeug bestätigt wiederrum die Abbestellung an die Wache und kehrt zurück.

Nachdem der Alarm am Einsatzort bearbeitet wurde, berichtet das zweite Fahrzeug an die Wache und übergibt dabei ein Einsatzprotokoll. Die Wache bestätigt den Bericht, so dass das Löschfahrzeug abrücken kann. Die Wache sendet das Einsatzprotokoll als nächstes zur Leitstelle woraufhin die Leitstelle das Protokoll bestätigt.

Als letztes archiviert die Leitstelle das Einsatzprotokoll. Der Notruf ist damit abgehandelt.

Name: Matrikelnummer:

Aufgabe 3 – Aktivitätsdiagramme (15 Punkte)

Sie sollen den allgemeinen Ablauf einer Alarmabhandlung bei der Feuerwehr als Aktivitätsdiagramm modellieren. Im Folgenden wird der zu modellierende Ablauf beschrieben:

Der Fahrzeugführer startet die Abhandlung des Alarms mit einer ersten Erkundung der Einsatzstelle. Danach meldet er seine Erkenntnisse der Leitstelle. Diese Erkenntnisse werden von der Leitstelle protokolliert und anschließend dem Fahrzeugführer bestätigt. Nachdem der Fahrzeugführer die Bestätigung der Leitstelle erhalten hat, beginnt er die Einsatzlage abzuschätzen. Dazu nutzt er sowohl seine eigenen Erkenntnisse als auch Informationen, die ihm die Leitstelle mitgegeben hat.

Hat ein Fahrzeugführer die Lage eingeschätzt, entscheidet er, ob er für den Einsatz seinen Feuerwehrtrupp benötigt oder nicht. Wird der Trupp benötigt, mobilisiert der Fahrzeugführer ihn. Während der Feuerwehrtrupp sich am Löschfahrzeug bereit macht, erkundet der Fahrzeugführer erneut den Einsatzort. Erst wenn die Erkundung beendet und der Feuerwehrtrupp bereit zum Einsatz ist, weist der Fahrzeugführer seinen Trupp ein. Dieser rückt daraufhin aus. Sobald der Trupp seine Aufgaben abgearbeitet hat, meldet er sich beim Fahrzeugführer zurück. Als nächstes berichtet der Fahrzeugführer der Leitstelle den aktuellen Stand der Alarmierung. Hat der Fahrzeugführer sich zuvor gegen den Einsatz seines Feuerwehrtrupps entschieden, berichtet er sofort der Leitstelle.

Nachdem der Fahrzeugführer der Leitstelle über den Stand der Alarmierung berichtet hat, quittiert die Leitstelle den Einsatz, während der Fahrzeugführer den Einsatz in seinen Unterlagen protokolliert und der Feuerwehrtrupp abrückt. Erst wenn der Einsatz vollständig quittiert, vom Fahrzeugführer protokolliert und der Abzug des Feuerwehrtrupps vollbracht wurde, beendet die Leitstelle den Einsatz.

Hinweis: Achten Sie beim Modellieren auf mögliche Akteure innerhalb des Aktivitätsdiagramms.

Name: Matrikelnummer:

Aufgabe 4 – Feature Diagramme (2+8+14+2+2 = 28 Punkte)

Ein Feature Diagramm ist genau dann konsistent, wenn es mindestens eine gültige Konfiguration hat, die nicht leer ist.

Teilaufgabe a) (2 Punkt)

Modellieren Sie ein Feature Diagramm, das nicht konsistent ist.

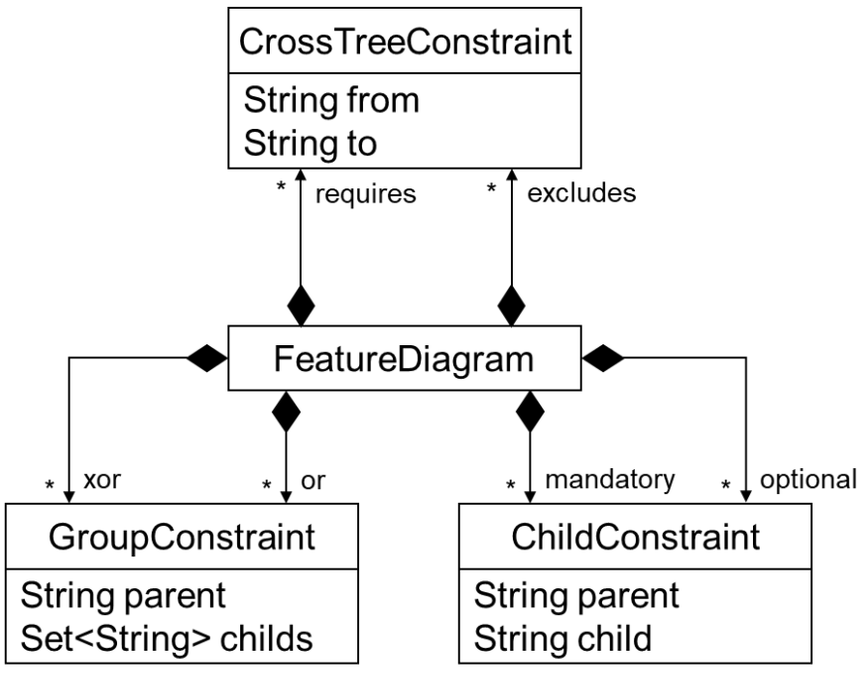
Teilaufgabe b) (8 Punkte)

Geben Sie für die folgenden Aussagen an, ob sie wahr oder falsch sind. Begründen Sie Ihre Antworten jeweils in minimal einem und maximal zehn Sätzen oder durch die Angabe eines passenden (Gegen-)Beispiels.

1) Wenn ein Feature Diagramm konsistent ist, dann kann das Löschen eines Requires- oder Excludes-Constraints in dem Feature Diagramm nicht zur Folge haben, dass das resultierende Feature Diagramm nicht konsistent ist (1 Punkt).
2) Wenn ein Feature Diagramm, das mindestens eine Xor-Gruppe enthält, nicht konsistent ist, dann ist jedes Feature Diagramm, das man erhält, wenn man in dem Feature Diagramm eine Xor-Gruppe zu einer Or-Gruppe ändert, auch inkonsistent (1 Punkt).
3) Jedes konsistente Feature Diagramm hat mindestens eine gütige Konfiguration, die das root-Feature des Feature Diagramms enthält (2 Punkte).
4) Jedes Feature Diagramm ohne Requires- und ohne Exlcudes-Constraints ist konsistent (2 Punkte).
5) Jedes Feature Diagramm ohne Xor-Gruppen und ohne Excludes-Constraints ist konsistent (2 Punkte).

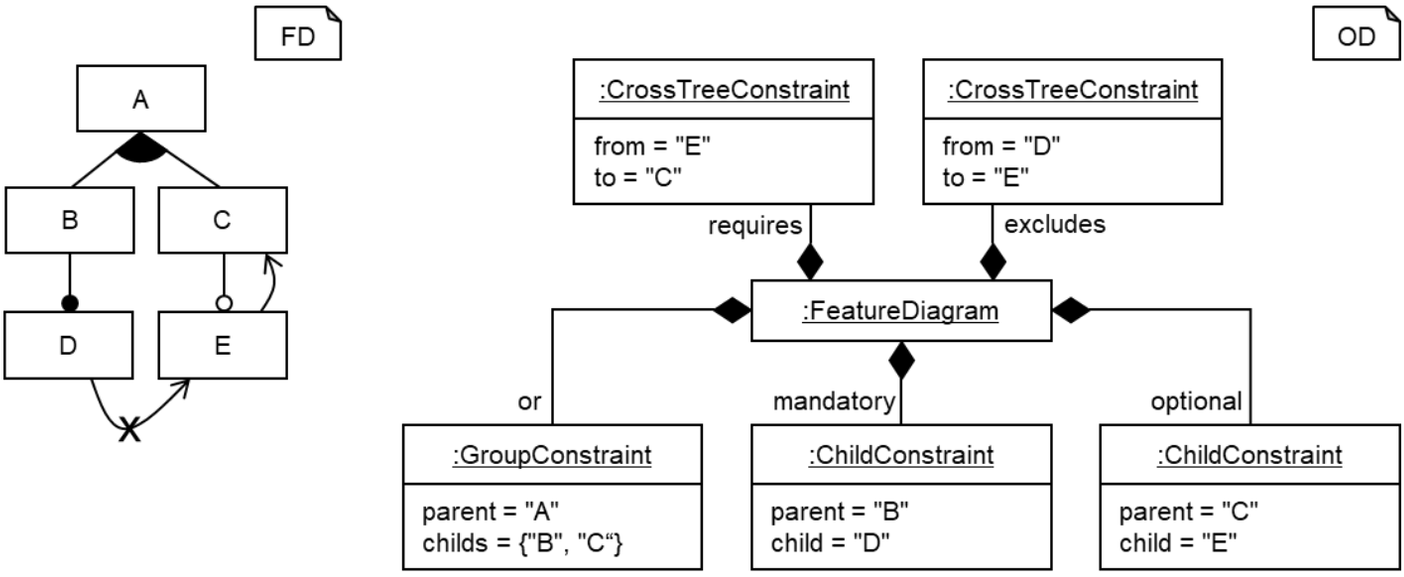
Teilaufgabe c) (14 Punkte)

Gegeben ist die im folgenden Klassendiagramm abgebildete Datenstruktur für Feature Diagramme.



Name: Matrikelnummer:

Die folgende Abbildung zeigt ein Feature Diagramm und ein Objektdiagramm, welches das Feature Diagramm in der obigen Datenstruktur repräsentiert.



Des Weiteren ist die folgende Methode `isValid` gegeben, die überprüft, ob eine Konfiguration `config` in einem Feature Diagramm `d` gültig ist. Konfigurationen werden durch Mengen von Strings repräsentiert. Jeder String in einer solchen Menge repräsentiert ein Feature.

Name: Matrikelnummer:

```
01  public boolean isValid(FeatureDiagram d, Set<String> config) {
02    for (ChildConstraint c : d.getMandatory()) {
03      if (!checkMandatoryChild(c, config)) {
04        return false;
05      }
06    }
07    for (ChildConstraint c : d.getOptional()) {
08      if (!checkOptionalChild(c, config)) {
09        return false;
10      }
11    }
12    for (CrossTreeConstraint c : d.getExcludes()) {
13      if (!checkExcludes(c, config)) {
14        return false;
15      }
16    }
17    for (CrossTreeConstraint c : d.getRequires()) {
18      if (!checkRequires(c, config)) {
19        return false;
20      }
21    }
22    for (GroupConstraint c : d.getOr()) {
23      if (!checkOr(c, config)) {
24        return false;
25      }
26    }
27    for (GroupConstraint c : d.getXor()) {
28      if (!checkXor(c, config)) {
29        return false;
30      }
31    }
32    return true;
33  }
```

Komplettieren Sie die im folgenden gezeigten Rümpfe der Methoden `checkMandatoryChild`, `checkOptionalChild`, `checkExcludes`, `checkRequires`, `checkOr` und `checkXor`. Die notwendigen Inhalte der Rümpfe können Sie sich aus der Definition von „konsistent“ für Feature Diagramme, der Datenstruktur und der Methode isValid herleiten. Sie können Pseudocode (oder auch Java oder eine andere Programmiersprache Ihrer Wahl) verwenden. Es reicht, wenn Sie in Ihrer Lösung ausschließlich die Inhalte der Rümpfe angeben. Sie müssen nicht das vollständig in der Aufgabenstellung gezeigte Gerüst abschreiben.

Name: Matrikelnummer:

```
/**
 * Gibt genau dann wahr zurück, wenn die Konfiguration config die Einschränkung des
 * Verpflichtenden-Kind-Constraints, der durch c repräsentiert  wird, erfüllt.
 * @param c ChildConstraint, der einen Verpflichtendes-Kind-Constraint repräsentiert
 * @param config Konfiguration
 * @return Wahr genau dann wenn config Bedingungen des Verpflichtendes-Kind-
 *         Constraints c erfüllt.
 */
private boolean checkMandatoryChild(ChildConstraint c, Set<String> config) {
  String parent = c.getParent();
  String child = c.getChild();









}
```

```
/**
 * Gibt genau dann wahr zurück, wenn die Konfiguration config die Einschränkungen
 * des Optionalen-Kind-Constraints, der durch c repräsentiert wird, erfüllt.
 * @param c ChildConstraint, der einen Optionales-Kind-Constraint repräsentiert
 * @param config Konfiguration
 * @return Wahr genau dann wenn config Bedingungen des Optionales-Kind-
 *         Constraints c erfüllt.
 */
private boolean checkOptionalChild(ChildConstraint c, Set<String> config) {
  String parent = c.getParent();
  String child = c.getChild();









}
```

Name: Matrikelnummer:

```
/**
 * Gibt genau dann wahr zurück, wenn die Konfiguration config die
 * Einschränkungen des Excludes-Constraints, der durch c repräsentiert wird,
 * erfüllt.
 * @param c CrossTreeConstraint, der einen Excludes-Constraint repräsentiert
 * @param config Konfiguration
 * @return Wahr genau dann wenn config Bedingungen des Excludes-
 *         Constraints c erfüllt.
 */
private boolean checkExcludes(CrossTreeConstraint c, Set<String> config) {
  String from = c.getFrom();
  String to = c.getTo();









}
```

```
/**
 * Gibt genau dann wahr zurück, wenn die Konfiguration config die
 * Einschränkungen des Requires-Constraints, der durch c repräsentiert
 * wird, erfüllt.
 * @param c CrossTreeConstraint, der einen Requires-Constraint repräsentiert
 * @param config Konfiguration
 * @return Wahr genau dann wenn config Bedingungen des requires Constraints
 *         c erfüllt.
 */
private boolean checkRequires(CrossTreeConstraint c, Set<String> config) {
  String from = c.getFrom();
  String to = c.getTo();









}
```

Name: Matrikelnummer:

```
/**
 * Gibt genau dann wahr zurück, wenn die Konfiguration config
 * die Einschränkung der Or-Gruppe, die durch c repräsentiert
 * wird, erfüllt.
 * @param c GroupConstraint, das eine Or-Gruppe repräsentiert
 * @param config Konfiguration
 * @return Wahr genau dann wenn config Bedingungen der Or-Gruppe
 *         c erfüllt
 */
private boolean checkOr(GroupConstraint c, Set<String> config) {
  String parent = c.getParent();
  Set<String> childs = c.getChilds();









}
```

Name: Matrikelnummer:

```
/**
 * Gibt genau dann wahr zurück, wenn die Konfiguration config
 * die Einschränkung der Xor-Gruppe, die durch c repräsentiert
 * wird, erfüllt.
 * @param c GroupConstraint, das eine Xor-Gruppe repräsentiert
 * @param config Konfiguration
 * @return Wahr genau dann wenn config Bedingungen der Xor-Gruppe
 *         c erfüllt
 */
private boolean checkXor(GroupConstraint c, Set<String> config) {
  String parent = c.getParent();
  Set<String> childs = c.getChilds();









}
```

Name: Matrikelnummer:

Teilaufgabe d) (2 Punkte)

Gegeben sind weiterhin die Datenstruktur und die Methoden aus Teilaufgabe c. Komplettieren Sie den Rumpf der im Folgenden gezeigten Methode isConsistent(FeatureDiagramm d), welche ein Feature Diagramm als Eingabe nimmt und genau dann wahr zurück gibt, wenn das Feature Diagramm d konsistent ist. Sie dürfen die Methoden aus Teilaufgabe c wiederverwenden.

```
/**
 * Gibt genau dann wahr zurück, wenn das Feature Diagramm d
 * konsistent ist.
 * @param d Feature Diagramm
 * @return Wahr genau dann wenn d konsistent ist
 */
public boolean isConsistent(FeatureDiagram d) {
  /* usedFeatures(d) gibt die Menge aller Features zurück,
     die im Feature Diagramm d benutzt werden.*/
  Set<String> features = usedFeatures(d);
  /* Sets.powerSet(s) gibt die Menge aller Teilmengen der
     Menge s zurück.*/
  for(Set<String> config : Sets.powerSet(features)) {









  }
  return false;
}
```

Teilaufgabe e) (2 Punkte)

Sei $n \geq 1$ eine positive ganze Zahl. Wie viele gültige Konfigurationen kann ein Feature Diagramm, das n Features beinhaltet, maximal haben? Geben Sie eine Formel in Abhängigkeit von n an. Beachten Sie für Ihre Antwort, dass für jedes n mindestens ein Feature Diagramm mit n Features existieren muss, das genau die Anzahl an gültigen Konfigurationen hat, die durch ihre Formel mit der Eingabe n definiert wird. Sie müssen Ihre Antwort nicht begründen.

Name: Matrikelnummer:

Aufgabe 5 – Testen (22 Punkte)

Die Methode `dijkstra` implementiert den Algorithmus von Dijkstra zur Lösung des Problems der Längen aller kürzesten Pfade in einem Graphen ausgehend von einem gegebenen Startknoten. Die Knoten des Graphen sind durch die ersten natürlichen Zahlen repräsentiert. Die Eingaben des Algorithmus sind ein zweidimensionales Array `g` und der Startknoten `n`. Der Wert `g[i][j]` repräsentiert das Gewicht der Kante vom Knoten `i` zum Knoten `j`. Falls `g[i][j] == null` gilt, dann existiert keine Kante von `i` nach `j`.

```
    public Optional<Integer[]> dijkstra(Integer[][] g, Integer n) {
01    if (g.length <= 0 || n >= g.length) {
02      return Optional.empty();
      }
03    for (int i = 0; i < g.length; i++) {
04      if (g[i] == null || g[i].length != g.length) {
05        return Optional.empty();
        }
06      for (int j = 0; j < g[i].length; j++) {
07        if (g[i][j] != null && g[i][j] < 0) {
08          return Optional.empty();
          }
        }
      }
09    Integer[] distance = new Integer[g.length];
10    for (int i = 0; i < g.length; i++) {
11      distance[i] = null;
      }
12    distance[n] = 0;
      Set<Integer> toProcess = new HashSet<>();
      IntStream.range(0, g.length).forEach(toProcess::add);
13    while (!toProcess.isEmpty()) {
14      Integer curDist = null, next = null;
15      for (Integer node : toProcess) {
16        if (distance[node] != null && (curDist == null || distance[node] < curDist)) {
17          next = node;
            curDist = distance[node];
          }
        }
18      toProcess.remove(next);
19      for (int i = 0; i < g[next].length; i++) {
20        if (toProcess.contains(i) && g[next][i] != null) {
21          if (distance[i] == null || (distance[next] + g[next][i] < distance[i])) {
22            distance[i] = distance[next] + g[next][i];
            }
          }
        }
      }
23    return Optional.of(distance);
    }
```

Name: Matrikelnummer:

Teilaufgabe a) (22 Punkte)

Konstruieren Sie einen Kontrollflussgraphen für die Methode `dijkstra`. Benutzen Sie die links vom Methodenrumpf angegebenen Nummern zur Beschriftung der zugehörigen Knoten im Kontrollflussgraphen.

Name: Matrikelnummer:

Aufgabe 6 – Muster (2+2+2+2+2 = 10 Punkte)

Teilaufgabe a) (2 Punkte)

Modellieren Sie ein Klassendiagramm mit den Elementen des „Adapter“ Musters. Beschreiben Sie anschließend in maximal drei Sätzen das Problem, das durch das Muster gelöst wird.

Teilaufgabe b) (2 Punkte)

Modellieren Sie ein Klassendiagramm mit den Elementen des „Koordinator“ Musters. Beschreiben Sie anschließend in maximal drei Sätzen das Problem, das durch das Muster gelöst wird.

Teilaufgabe c) (2 Punkte)

Modellieren Sie ein Klassendiagramm mit den Elementen des „Wechselnde Rolle“ Musters. Beschreiben Sie anschließend in maximal drei Sätzen das Problem, das durch das Muster gelöst wird.

Teilaufgabe d) (2 Punkte)

Modellieren Sie ein Klassendiagramm mit den Elementen des „Gruppenhistorie“ Musters. Beschreiben Sie anschließend in maximal drei Sätzen das Problem, das durch das Muster gelöst wird.

Teilaufgabe e) (2 Punkte)

Modellieren Sie ein Klassendiagramm mit den Elementen des „Composite“ Musters. Beschreiben Sie anschließend in maximal drei Sätzen das Problem, das durch das Muster gelöst wird.