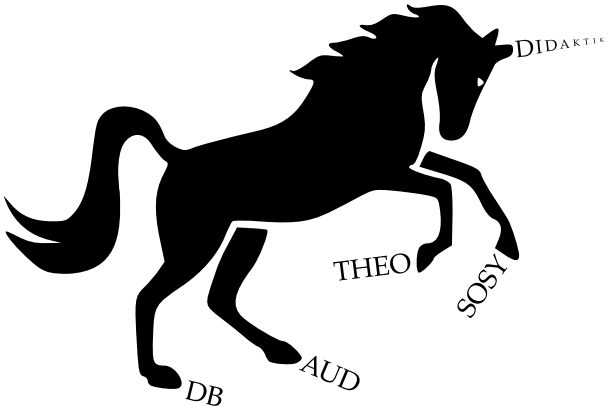
Die komplette Sammlung

Alle Aufgaben



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Inhaltsverzeichnis

Teil I

Softwaresysteme (SOSY)

EXtreme Programming
V-Modell
Wasserfallmodell
SCRUM
Prototyping
Unit-Test
Anforderungsanalyse

Projektmanagement

Allgemeine Software-Technologie, Vorgehensmodelle und Requirements Engineering

Kreuzen Sie bei der folgenden Multiple-Choice-Frage die richtige(n) Antwort(en) an. Auf falsch gesetzte Kreuze gibt es je einen Minuspunkt. Die Aufgabe wird nicht mit weniger als 0 Punkten gewertet.

(a) Welche Vorgehensmodelle sind für Projekte mit häufigen Änderungen gedacht?

- ☐ EXtreme Programming (XP)
- ☐ Das V-Modell 97
- ☐ Wasserfallmodell
- ☐ Scrum

Lösungsvorschlag

- ☒ EXtreme Programming (XP)
- ☐ Das V-Modell 97
- ☐ Wasserfallmodell
- ☒ Scrum

(b) Welche der folgenden Aussagen ist korrekt?

- ☐ Mittels Prototyping versucht man die Anzahl an nötigen Unit-Tests zu reduzieren.
- ☐ Ein Ziel von Prototyping ist die Erhöhung der Qualität während der Anforderungsanalyse.
- ☐ Mit Prototyping versucht man sehr früh Feedback von Stakeholdern zu erhalten.

Lösungsvorschlag

- ☐ Mittels Prototyping versucht man die Anzahl an nötigen Unit-Tests zu reduzieren.
- ☒ Ein Ziel von Prototyping ist die Erhöhung der Qualität während der Anforderungsanalyse.
- ☒ Mit Prototyping versucht man sehr früh Feedback von Stakeholdern zu erhalten.

(c) Welche der folgenden Aussagen ist korrekt?

- ☐ Das Wasserfallmodell sollte nur für große Projekte eingesetzt werden, da der Einarbeitungsaufwand sehr groß ist.
- ☐ Eine gute Anforderungsspezifikation muss vor allem für Ingenieure verständlich sein, da die Anforderungsspezifikation die Grundlage der Systementwicklung bildet.
- ☐ Verifikation ist der Prozess der Beurteilung eines Systems mit dem Ziel festzustellen, ob die spezifizierten Anforderungen erfüllt sind.
- ☐ Durch Validierung kann überprüft werden, ob das Produkt den Erwartungen des Kunden entspricht.
- ☐ Mit Hilfe eines Black-Box-Tests kann man die Korrektheit eines Programmcodes beweisen.

Lösungsvorschlag

- ☐ Das Wasserfallmodell sollte nur für große Projekte eingesetzt werden, da der Einarbeitungsaufwand sehr groß ist.
- ☐ Eine gute Anforderungsspezifikation muss vor allem für Ingenieure verständlich sein, da die Anforderungsspezifikation die Grundlage der Systementwicklung bildet.
- ☐ Verifikation ist der Prozess der Beurteilung eines Systems mit dem Ziel festzustellen, ob die spezifizierten Anforderungen erfüllt sind.
- ☒ Durch Validierung kann überprüft werden, ob das Produkt den Erwartungen des Kunden entspricht.
- ☐ Mit Hilfe eines Black-Box-Tests kann man die Korrektheit eines Programmcodes beweisen.

Use Case

Das Entwicklerteam von Teacher-Data hat einen neuen Auftrag bekommen. Sie sollen für die Automaten-Videothek DVDRental die Software entwickeln. Nach den ersten Gesprächen konnten folgende Informationen zusammengetragen werden:

- DVD-Verleih-Automat:
  - Der Kunde soll einfach, schnell und günstig am Automaten 24 Stunden, sieben Tage die Woche DVDs ausleihen können. Der Kunde hat des Weiteren die Möglichkeit, Filme vorab über das Internet zu reservieren. In der DVD-Verleihstation gibt es drei verschiedene Automaten (siehe Zeichnung).
  - Um DVDs ausleihen zu können, müssen sich Neukunden während der Service- und Anmeldezeiten der DVD-Verleihstation bei einem Mitarbeiter anmelden. Dabei werden die benötigten Daten (*Name*, *Vorname*, *Geburtsdatum*, *Anschrift*, *Personalausweisnummer*, *Guthaben*, *Kennwort*) des Kunden aufgenommen und im System gespeichert. Zur persönlichen Identifikation des Kunden wird anschließend der Fingerabdruck am Automaten eingelesen. Der

Kunde bekommt eine Magnetkarte ausgehändigt, um sich an den Automaten anmelden zu können.

- Beim DVD-Ausleih muss sich der Kunde mit der Magnetkarte und seinem Fingerabdruck am Automaten für die Filmauswahl identifizieren. Am Eingabe-Terminal hat der Kunde die Möglichkeit Filme nach *Genre*, *Schauspieler*, *Titel*, *TOP-10*, *Neuerscheinungen* und *Stichworten* zu suchen. Suchergebnisse werden mit folgenden Informationen auf dem Display präsentiert: *Titel des Films*, *Regisseur*, *Hauptdarsteller*, *Kurzbeschreibung*, *Erscheinungsjahr*, *Altersfreigabe* und *Verfügbarkeit*. Nach der Wahl eines Films, erhält der Kunde seine Magnetkarte zurück und kann den Film am Automaten für die Ausund Rückgabe nach erneuter Identifikation durch die Karte und den Fingerabdruck abholen. Die DVD wird über den Ausgabeschacht ausgegeben.
- Bei der Rückgabe einer DVD muss sich der Kunde am Automaten für die Aus- und Rückgabe von Filmen mit der Magnetkarte und seinem Fingerabdruck am Automaten identifizieren. Nach erfolgreicher Identifikation führt der Kunde die DVD in den Rückgabeschacht ein. Der Automat liest daraufhin den Barcode auf der DVD und sortiert diese eigenständig wieder ein und bucht die Leihgebühr vom Guthaben auf der Karte ab.
- Das Guthaben kann am Automaten zum Aufladen des Guthabens aufgeladen werden. Nach Eingabe der Kundenkarte können Münzen (5, 10, 20, 50 Cent, 1 und 2€) und Scheine (5, 10, 20, 50€) bar eingezahlt werden. Bei einem Betrag von 20€ gibt es einen Bonus von 3€, bei 30€ einen von 5€ und bei einem von 50€ von 10€. Die Karte wird bei beendeter Geldeingabe nach Drücken des roten Knopfes wieder ausgeworfen.
- Reservierung von Filmen über das Internet: Nach erfolgreicher Anmeldung mit Kundennummer und Geheimzahl auf der betreibereigenen Homepage kann der Kunde nach *Genre*, *Schauspieler*, *Titel*, *TOP-10*, *Neuerscheinungen* und *Stichworten* Filme suchen. Suchergebnisse werden mit folgenden Informationen präsentiert: *Titel des Films*, *Regisseur*, *Hauptdarsteller*, *Kurzbeschreibung*, *Erscheinungsjahr*, *Altersfreigabe* und *Verfügbarkeit*. Der Kunde kann nach der Suche einzelne Filme reservieren, die er innerhalb von zwei Stunden abholen muss. Sollte dies nicht in dem Zeitraum geschehen, so werden die Filme nach zwei Stunden wieder zum Ausleihen freigegeben.

- Weitere Rahmenbedingungen:

  - Wird die Kundenkarte im Automaten vergessen, so wird sie zur Sicherheit automatisch eingezogen und kann während der Servicezeiten wieder abgeholt werden.
  - Die maximale Leihdauer beträgt 10 Tage.
  - Hat der Kunde versehentlich den falschen Film ausgeliehen, so kann er diese bis 10 Minuten nach Ausgabe der DVD ohne Berechnung einer Leihgebühr wieder am Automaten zurückgeben.
  - Die Leihgebühr für jeden angefangenen Tag (24h) beträgt 2€.

- Zu jedem Film gibt es mehrere Exemplare.
- Die Ausgabe und Einsortierung von DVDs übernimmt ein DVD-Archiv-Roboter, der über eine Schnittstelle angesteuert werden muss.

Nicht-funktionale Anforderungen
Funktionale Anforderungen

(a) Identifizieren Sie die Aktoren des oben beschriebenen Systems.

Lösungsvorschlag

Kunde, Service-Mitarbeiter, Wartung, DVD-Archiv-Roboter

(b) Identifizieren Sie die nicht-funktionalen und funktionalen Anforderungen (als Use Cases).

Lösungsvorschlag

Nicht-funktional: Bedienoberfläche Web / Automat gleich, einfache Handhabbarkeit, schnell, Dauerbetrieb, Ausfallquote, Leihdauer

Funktional: Geldkarte aufladen, Film suchen, Film auswählen, Film entnehmen, Film zurückgeben, Kunden aufnehmen, Film reservieren

(c) Geben Sie eine formale Beschreibung für drei Use Cases an. Orientieren können Sie sich an folgendem Beispiel:

Lösungsvorschlag

Geldkarte aufladen

Use Case:	Geldkarte aufladen
Ziel:	Auf der Geldkarte befindet sich ein gewünschter Betrag an Geld.
Kategorie:	primär
Vorbedingung:	Geldkarte befindet sich im Automaten.
Nachbedingung Erfolg:	Auf der Geldkarte befindet sich der gewünschte Betrag.
Nachbedingung Fehlschlag:	Betrag auf der Karte ist der gleiche wie davor → Fehlermeldung
Akteure:	Kunde
Auslösendes Ereignis:	Geldkarte wird in den Automaten zum Aufladen des Guthabens geschoben.
Beschreibung:	(i) Kunde schiebt Karte in den Automaten. (ii) Kunde wirft Münzen oder gibt einen Geldschein ein. (iii) Kunde drückt Knopf zum Auswerfen der Karte. (iv) Karte wird ausgegeben.
Erweiterungen:	Schein oder Münze wird nicht akzeptiert. → Fehlermeldung
Alternativen:	Mindest-Guthaben auf der Karte

Film suchen

Use Case:	Film suchen
Ziel:	Anzeigen des gesuchten Films
Kategorie:	primär
Vorbedingung:	Kunde hat sich mit Magnetkarte und Fingerabdruck identifiziert.
Nachbedingung Erfolg:	Eine Liste von Filmen wird angezeigt.
Nachbedingung Fehlschlag:	Kein Film gefunden → Fehlermeldung
Akteure:	Kunde
Auslösendes Ereignis:	Kunde will einen Film ausleihen.
Beschreibung:	Kunde gibt Genre, Schauspieler, Titel oder Stichwort ein oder lässt sich Neuerscheinungen und TOP10 auflisten.
Erweiterungen:	–
Alternativen:	Vorgaben von weiteren Suchkriterien

Kunden aufnehmen

Use Case:	Kunden aufnehmen
Ziel:	Kunde ist in der Kundendatei.
Kategorie:	primär
Vorbedingung:	Kunde ist noch nicht in der Kundendatei vorhanden.
Nachbedingung Erfolg:	Kunde ist als Mitglied in der Kundendatei aufgenommen.
Nachbedingung Fehlschlag:	Kunde kann nicht in die Datei aufgenommen werden.
Akteure:	Kunde, Service-Mitarbeiter
Auslösendes Ereignis:	Kunde stellt zu den Öffnungszeiten bei einem Service-Mitarbeiter einen Mitgliedsantrag.
Beschreibung:	(i) Kunde meldet sich zu den Öffnungszeiten bei einem Service-Mitarbeiter an. (ii) Service-Mitarbeiter erfasst nötige Daten (Name, Vorname, Geburtsdatum, Anschrift, Personalausweisnummer, Kennwort). (iii) Service-Mitarbeiter kassiert einen Startbetrag und erfasst ihn bei den Kundendaten. (iv) Der Fingerabdruck des Kunden wird erfasst und gespeichert. (v) Kunde bekommt eine Magnetkarte mit seinen Daten.
Erweiterungen:	-
Alternativen:	Kunde ist bereits im System und lässt nur die Kundendaten ändern.

(d) Verfeinern Sie nun die Use Cases, indem Sie ähnliche Teile identifizieren und daraus weitere Use Cases bilden.

Lösungsvorschlag

Identifizierung des Kunden an den Automaten, Filmsuche

(e) Nennen Sie mindestens drei offene Fragen, die in einem weiteren Gespräch mit dem Kunden noch geklärt werden müssten.[1]

[1] Quelle: Universität Stuttgart, Institut für Automatisierungs- und Softwaretechnik Prof. Dr.-Ing. Dr. h. c. P. Göhner

Lösungsvorschlag Wasserfallmodell

- Was passiert nach der maximlaen Leihdauer?
- Was passiert, wenn der Finger nicht zur Karte passt?
- Was passiert mit defekten DVDs?
- Was passiert bei Rückgabe der falschen DVD?

Wasserfallmodell

Die Firma Teacher-Data soll für einen Auftraggeber ein Informationssystem entwickeln. Das Entwicklerteam entscheidet sich, nach dem *Wasserfallmodell* vorzugehen.

(a) Geben Sie an, welche *Phasen* dabei durchlaufen werden und erläutern Sie kurz deren *Zweck / Intention / Aufgabe*.

Lösungsvorschlag

(i) Problem- und Systemanalyse

Grobe Formulierung der Ziele (z. B. Einsatzplattform Windows, einheitliche Software zur Unterstützung des Verkaufs, ...)

- Erfassung des Problembereichs (z. B. Interaktion zwischen Händler und Hersteller soll komplett über das Netz funktionieren.)
- Machbarkeitsstudie (z. B. Händler sind bereit, sich an das zu entwickelnde System anzupassen.)
- Analyse des Ist-Zustandes (z. B. Händler A bietet Interface X wohingegen Händler B Interface Z bietet. Es wird bereits teilweise bei Händler C über das Netz bestellt. ...)
- Anforderungsspezifikation (Lastenheft) (z. B. Einheitliche Oberfläche für alle Benutzer beim Hersteller, möglichst wenige Veränderungen für die Mitarbeiter (= schnelle Eingewöhnungsphase) ...)
- Systemspezifikation (Pflichtenheft) (z. B. soll der Ablauf eines Kaufvorgangs folgendermaßen aussehen: Händlerauswahl, Bestellung, Abrechnung ... Die Benutzeroberfläche soll folgende Informationen anzeigen: Lieferstatus, ...)

(ii) Systementwurf

- Systemarchitektur (z. B. Klassendiagramm, Festlegung von Komponenten ...)
- Schnittstellen der Komponenten festlegen (z. B. Datenbank, Middleware, ...)
- Modulkonzept (z. B. Modulstruktur und Art der Module (funktional, prozedural, objektorientiert)
- Definition der Einzelschnittstellen (z. B. Interfaces und abstrakte Klassen in Java)

(iii) Implementierung

- Strukturierung der Komponenten in Module (z. B. Klassenrümpfe)
- Spezifikation einzelner Module (z. B. exakte Beschreibung eines Moduls)
- Codierung, Generierung, Wiederverwendung einzelner Module (z. B. Methoden in Klassen, ...)

(iv) Integration und Test

- Integration von Modulen (z. B. Zusammensetzen von Datenbank und Applikation zum Server)
- Integration von Komponenten (z. B. Einsetzen von Client und Server ins Gesamtsystem.)
- Testen und Verifikation (z. B. Testen, ob die Implementierung zusammen mit der Umgebung funktioniert. Zusichern, dass gewisse Eigenschaften gültig sind.)

(v) Wartung und Weiterentwicklung

- Wartung (z. B. Korrektur eines Fehlers im Programm, Portierung auf anderes Betriebssystem, ...)
- Erweitern der Funktionalität (z. B. Einbeziehen einer neuen Option im Geschäftsprozess)
- Anpassung der Funktionalität (z. B. Anpassung an Änderung im Geschäftsprozess)

(b) Während des Entwicklungsprozesses werden nach und nach im Folgenden aufgelistete Teilprodukte entstehen. Ordnen Sie diese den jeweiligen Phasen zu.

- Schätzung der Entwicklungskosten
- Dokumentation der Änderungen nach Produktabnahme
- Pflichtenheft
- Beschreibung der Datenstruktur in der Datenbank
- Integration von Modulen/Komponenten
- Betriebsbereites Informationssystem
- Beschreibung der Schnittstelle einzelner Softwarekomponenten
- Quellcode für die GUI
- Durchführbarkeitsstudie
- Systemarchitektur

Lösungsvorschlag

- Schätzung der Entwicklungskosten → *Anforderungsdefinition*
- Dokumentation der Änderungen nach Produktabnahme → *Betrieb und Wartung*
- Pflichtenheft → *Anforderungsdefinition*

- Beschreibung der Datenstruktur in der Datenbank → *System- und Softwareentwurf*
- Integration von Modulen/Komponenten → *Integration und Systemtest*
- Betriebsbereites Informationssystem → *Betrieb und Wartung*
- Beschreibung der Schnittstelle einzelner Softwarekomponenten → *System- und Softwareentwurf*
- Quellcode für die GUI → *Implementierung und Komponententest*
- Durchführbarkeitsstudie → *Anforderungsdefinition*
- Systemarchitektur → *System- und Softwareentwurf*

Vorgehensmodelle

Dem Team von Teacher-Data kommen Zweifel, ob die Entscheidung für das Wasserfallmodell für die Umsetzung des Informationssystems richtig war, oder ob ein anderes Vorgehen eventuell besser wäre. Beurteilen und vergleichen Sie die Vorgehensmodelle

- Wasserfallmodell
- Spiralmodell
- V-Modell
- Evolutionäres Modell oder Inkrementelles Modell
- agile Entwicklung, wie z. B. Scrum

nach den folgenden Gesichtspunkten:

(a) Größe des Entwicklerteams

(b) Komplexität des Projekts

(c) Bekanntheit der Anforderungen

(d) Änderung der Anforderungen

(e) Zeitspielraum (Time-to-Market; was muss bis wann fertig sein?)

(f) Dokumentation

(g) IT-Kenntnisse des Kunden

(h) Durchschnittliche Anzahl an Iterationen

Lösungsvorschlag

White-Box-Testing
Black-Box-Testing

	Wasserfallmodell	Spiralmodell	V-Modell	evolutionär / inkrementell	agil
Größe des Entwicklerteams	diskutiert die Projektgröße nicht	mittlere Teams	mittlere und große Teams	diskutiert die Projektgröße nicht	ca. 3-9 Entwickler (ohne Product Owner und Scrum Master)
Komplexität des Projekts	einfache Projekte mit stabilen Anforderungen	einfache Projekte mit stabilen Anforderungen; Anforderungen können jedoch in den Iterationen angepasst werden.	komplexe Projekte; bei einfachen Projekten relativ viel bürokratischer Overhead.	große, lange und komplexe Projekte	hohe Komplexität möglich
Bekanntheit der Anforderungen	Alle Anforderungen müssen zu Beginn des Projekts bekannt sein.	Es muss eine initiale Menge von Anforderung bekannt sein. Der Kunde kann die Anforderungen aber aufgrund von Prototypen erweitern.	evolutionär: Alle Anforderungen müssen zu Beginn des Projekts bekannt sein. inkrementell: Anforderungen müssen nicht von Anfang an bekannt sein.	Anforderungen müssen von Anfang an bekannt sein.	Es muss eine initiale Menge von Anforderung bekannt sein. Der Kunde kann die Anforderungen aber laufend erweitern / ggf. ändern.
Änderungen der Anforderungen	Schwer möglich. Durch den strikten Top-Down-Ansatz von der Anforderung hin zum Code zieht eine Änderung einen tiefen Eingriff in den Prozess mit sich.	Gut möglich. In jeder Runde der Spirale können die Anforderungen neu definiert werden.	Schwer möglich → strikter Top-Down-Ansatz	evolutionär: häufige Änderungen möglich. inkrementell: Änderungen teilweise möglich.	rasche Anpassung an neue Anforderungen möglich (deshalb agil!)
Zeitspielraum	Wenig Spielraum. Üblicherweise muss der ganze Prozess durchlaufen werden. Es ist höchstens möglich, den Prozess zu beschleunigen, indem man Anforderungen streicht. Es wird also alles zum Schluss fertig.	Relativ flexibel. Sobald ein Prototyp existiert, der akzeptabel ist, kann dieser auf den Markt gebracht werden.	Wenig Spielraum (vgl. Wasserfallmodell)	Einsatzfähige Produkte in kurzen Zeitabständen, daher relativ flexibel.	Unterteilung in Sprints, daher schnell lauffähige Prototyoen vorhanden. Bei der Planung des nächsten Sprints kann auf neue zeitliche Gegebenheiten relativ flexibel eingegangen werden.
Dokumentation	Viel Dokumentation – Für jede Phase wird eine komplette Dokumentation erstellt.	Sehr viel Dokumentation – In jedem Zyklus werden alle Phasen (inkl. Dokumentation) durchlaufen, aber nur im Umfang des Prototypen. Es werden alle Artefakte und Tätigkeiten dokumentiert.	Viel Dokumentation – Gerade in den ersten Phasen wird sehr viel über die Software schriftlich festgehalten. Aber auch für die anderen Phasen wird eine komplette Dokumentation erstellt.		Anforderungsdokumentation ist sehr wichtig. Ansonsten ist funktionierende Software höher zu bewerten als eine umfangreiche Dokumentation.
IT-Kenntnisse des Kunden	Wenig Kenntnisse nötig. Meistens schreibt der Kunde das Lastenheft und der Entwickler versucht dieses dann in einem Pflichtenheft umzusetzen. Wenn das Lastenheft vom Entwickler als machbar befunden wird, bekommt der Kunde das fertige Produkt.	Wenig Kenntnisse nötig. – Kunde sieht immer nur die fertigen Prototypen und äußert dann seine Wünsche.	Wenig Kenntnisse nötig (vgl. Wasserfallmodell)	Wenig Kenntnisse nötig	Wenig Kenntnisse nötig, aber von Vorteil, da enge Zusammenarbeit mit dem Kunden
Durchschnittliche Anzahl an Iterationen	vgl. V-Modell	ca. 3-5 Iterationen	Nur lange Zyklen. – Mit dem V-Modell sind nur sehr lange Zyklen handhabbar, da immer wieder der ganze Entwicklungsprozess durchlaufen werden muss.		variable Anzahl an Sprints, je nach Projektgröße

Aufgabe 1: Grundwissen

(a) Nennen Sie die wesentlichen Unterschiede zwischen *White-Box-Testen* und *Black-Box-Testen* und geben Sie jeweils zwei Beispiele an.

Lösungsvorschlag

Funktionalorienteres Testen
V-Modell
Agile Methoden

White-Box-Tests sind *strukturorientiert*, z. B. Kontrollflussorientiertes Testen oder Datenflussorientiertes Testen.

Black-Box-Test sind *spezifikations- und funktionsorientiert* (nur Ein- und Ausgabe relevant), z. B. Äquivalenzklassenbildung oder Grenzwertanalyse.

(b) Geben Sie drei nicht-funktionalorientierte Testarten an.

Lösungsvorschlag

(i) Performanztest
(ii) Lasttest
(iii) Stresstest

(c) Nennen Sie die vier verschiedenen Teststufen aus dem V-Modell und erläutern Sie deren Ziele.

Lösungsvorschlag

Komponenten-Test: Fehlerzustände in Modulen finden.

Integrations-Test: Fehlerzustände in Schnittstellen und Interaktionen finden.

System-Test: Abgleich mit Spezifikation.

Abnahme-Test: Vertrauen in System und nicht-funktionale Eigenschaften gewinnen.

(d) Nennen Sie fünf Aktivitäten des Testprozesses.

Lösungsvorschlag

(i) Testplanung und Steuerung,
(ii) Testanalyse und Testentwurf,
(iii) Testrealisierung und Testdurchführung,
(iv) Bewertung von Endekriterien und Bericht,
(v) Abschluss der Testaktivitäten

46116 / 2013 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 2

Erläutern Sie in jeweils ca. 3 Sätzen die folgenden Begriffe aus der modernen Softwaretechnologie: *Pair Programming*, *Refactoring*, *agile Softwareentwicklung*.

Lösungsvorschlag

Pair Programming Beim Pair Programming entwickeln zwei Personen die Software zusammen. Es gibt einen sogenannten Driver - die Person, die tippt - und einen sogenannten Navigator - die Person, die kritisch den Code überprüft. Das Pair Programming ist wichtiger in agilen Entwicklungsmethoden wie z. B dem Exterme Programming (XP).

Refactoring Unter Refactoring versteht man die Verbesserung der Code- und Systemstruktur mit dem Ziel einer besseren Wartbarkeit. Dabei sollen Lesbarkeit, Verständlichkeit, Wartbarkeit und Erweiterbarkeit verbessert werden, mit dem Ziel, den jeweiligen Aufwand für Fehleranalyse und funktionale Erweiterungen deutlich zu senken. Im Refactoring werden keine neuen Funktionalitäten programmiert.

agile Softwareentwicklung Agile Entwicklungsprozesse gehen auf das Agile Manifest, das im Jahr 2001 veröffentlicht wurde, zurück. Dieses Agile Manifest bildet die Basis für agile Entwicklungsprozesse wie eXtreme Programming oder SCRUM. Das Manifest beinhaltet 12 Grundprinzipien für die moderne agile Software-Entwicklung. Hauptzielsetzung ist, rasch eine funktionsfähige Software-Lösung auszuliefern, um den Kundenwunsch bestmöglich zu erfüllen. Dabei spielt die Interaktion mit dem Kunden eine zentrale Rolle.

46116 / 2014 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 1

Aufgabe 1: Allgemeine SWT, Vorgehensmodelle und Requirements Engineering

Kreuzen Sie für die folgenden Multiple-Choice-Fragen genau die richtigen Antworten deutlich an. Es kann mehr als eine Antwort richtig sein.

Jedes korrekt gesetzte oder korrekt nicht gesetzte Kreuz wird mit 1 Punkt gewertet. Jedes falsch gesetzte oder falsch nicht gesetzte Kreuz wird mit -1 Punkt gewertet. Eine Frage kann entwertet werden, dann wird sie nicht in der Korrektur berücksichtigt. Einzelne Antworten können nicht entwertet werden. Entwerten Sie eine Frage wie folgt

Die gesamte Aufgabe wird nicht mit weniger als 0 Punkten gewertet.

(a) Welche Aussage ist wahr?

- ☐ Je früher ein Fehler entdeckt wird, umso teurer ist seine Korrektur.
- ☐ Je später ein Fehler entdeckt wird, umso teurer ist seine Korrektur.
- ☐ Der Zeitpunkt der Entdeckung hat keinen Einfluss auf die Kosten.

Lösungsvorschlag

2 ist richtig: Je später der Fehler entdeckt wird, desto mehr wurde er schon in das Projekt „eingearbeitet“, daher dauert das Beseitigen des Fehlers länger und das kostet mehr Geld.

(b) Mit welcher Methodik können Funktionen spezifiziert werden?

- ☐ Als Funktionsvereinbarung in einer Programmiersprache
- ☐ Mit den Vor- und Nachbedingungen von Kontrakten
- ☐ Als Zustandsautomaten

Lösungsvorschlag

2 und 3 ist richtig: Die Spezifikation soll unabhängig von einer Programmiersprache sein.

(c) Welche Vorgehensmodelle sind für Projekte mit häufigen Änderungen geeignet?

- ☐ Extreme Programming (XP)
- ☐ Das V-Modell 97
- ☐ Scrum

Lösungsvorschlag

1 und 3 ist richtig. Das V-Modell ist ein starres Vorgehensmodell, bei dem alle Anforderungen zu Beginn vorhanden sein müssen.

(d) Welche der folgenden Aussagen ist korrekt?

- ☐ Mittels Prototyping versucht man die Anzahl an nötigen Unit-Tests zu reduzieren.
- ☐ Ein Ziel von Prototyping ist die Erhöhung der Qualität während der Anforderungsanalyse.
- ☐ Mit Prototyping versucht man sehr früh Feedback von Stakeholdern zu erhalten.

Lösungsvorschlag

2 und 3 ist richtig: Prototypen müssen auch getestet werden. Es kann nicht an Tests gespart werden. Durch das häufige Feedback des Kunden / der Stakeholder können die Anforderungen immer genauer und klarer erfasst werden.

(e) Welche der folgenden Aussagen ist korrekt?

- ☐ Bei der Architektur sollten funktionale und nicht-funktionale Anforderungen beachtet werden.
- ☐ Bei der Architektur soliten nur funktionale Anforderungen beachtet werden.
- ☐ Bei der Architektur sollten nur nicht-funktionale Anforderungen beachtet werden.
- ☐ Bei der Architektur sollte auf die mögliche Änderungen von Komponenten geachtet werden.

Lösungsvorschlag

1 und 4 ist richtig: Mögliche Änderungen werden durch klar definierte Schnittstellen und wenig Kopplung der Komponenten erleichtert. (Kopplung handelt von Abhängigkeiten zwischen Modulen. Kohäsion handelt von Abhängigkeiten zwischen Funktionen innerhalb eines Moduls.)

Prozessmodelle
Wasserfallmodell

46116 / 2014 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 2

Software wird oft in definierten Prozessen entwickelt. Diese nennt man Vorgehensmodelle.

Allgemein

(a) Was sind die Aufgaben eines Vorgehensmodells im Allgemeinen?

Lösungsvorschlag

- liefert Erfahrungen und bewährte Methoden
- beschreibt die am Projekt beteiligten Rollen
- legt Aufgaben und Aktivitäten fest
- definiert einheitliche Begriffe
- gibt Techniken, Werkzeuge, Richtlinien / Standards an

(b) Was sind die wesentlichen Bestandteile eines Vorgehensmodells und in welcher Beziehung stehen diese zueinander?

Lösungsvorschlag

Bestandteile: Anforderungsanalyse, Modellierung, Implementierung, Test, Auslieferung, Wartung

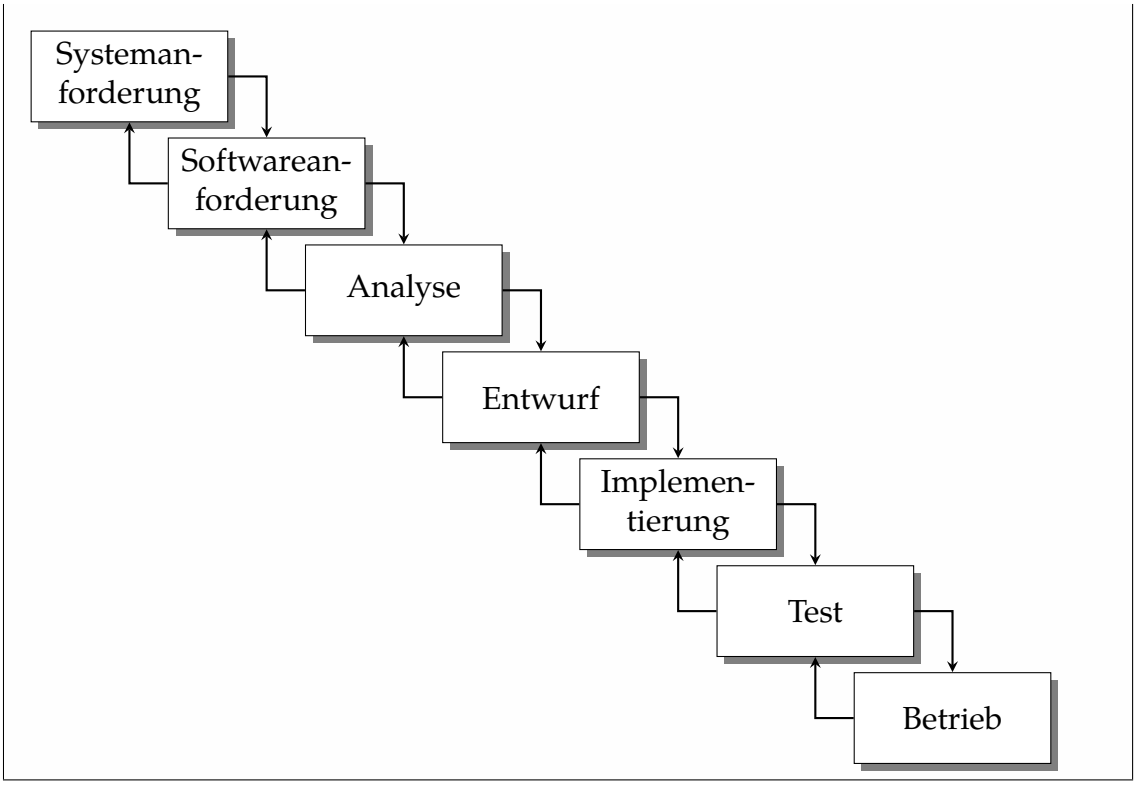
Die einzelnen Phasen bauen immer aufeinander aus. Je nach Vorgehensmodell können sich Phasen auch wiederholen (Prototyping, Scrum...).

Ein frühes Vorgehensmodell ist das von Dr. Winston Royce 1970 formalisierte Wasserfallmodell.

(a) Geben Sie eine schematische Darstellung des Wasserfallmodells an.

Lösungsvorschlag

V-Modell



(b) Nennen Sie zwei Probleme des Modells und erläutern Sie diese kurz.

Lösungsvorschlag

Fehler werden ggf. erst am Ende des Entwicklungsprozesses erkannt, da erst dort das Testen stattfindet. Dadurch kann die Behebung eines Fehlers sehr aufwändig und somit teuer werden.

Der Kunde / Endanwender wird erst nach der Implementierung wieder eingebunden. Das bedeutet, dass er nach der Stellung der Anforderungen keinen Einblick mehr in den Prozess hat und somit auch nicht gegensteuern kann, falls ihm etwas nicht gefällt oder er etwas nicht bedacht hat.

(c) In welchen Situationen lässt sich das Wasserfallmodell gut einsetzen?

Lösungsvorschlag

Das Wasserfallmodell ist geeignet, wenn es sich um ein von Anfang an klar definiertes Projekt ohne große Komplexität handelt, bei dem alle Anforderungen, Aufwand und Kosten schon zu Beginn des Projekts feststehen bzw. abgeschätzt werden können.

Barry Boehm erweiterte das Wasserfallmodell 1979 zum so genannten V-Modell.

(a) Geben Sie eine schematische Darstellung des V-Modells an.

(b) Nennen Sie zwei Probleme des Modells und erläutern Sie diese kurz.

Lösungsvorschlag

- Die Nachteile des Wasserfallmodells bestehen weiterhin!
- Nicht für kleine Projekte geeignet, da aufwändige Tests vorgesehen sind, die im Kleinen detailliert meist nicht stattfinden (können).

(c) Welchen Vorteil hat das V-Modell gegenüber dem Wasserfallmodell?

Lösungsvorschlag

Für jedes Dokument besteht ein entsprechender Test (Validierung / Verifikation). Dabei kann die Planung der Tests schon vor der eigentlichen Durchführung geschehen, so dass Aktivitäten im Projektteam parallelisiert werden können. So kann zum Beispiel der Tester die Testfälle für den Akzeptanztest (=Test des Systementwurfs) entwicklen, auch wenn noch keine Implementierung existiert.

In neuerer Zeit finden immer häufiger iterative und inkrementelle Vorgehensweisen Anwendung.

(a) Erklären Sie den Begriff iterative Softwareentwicklung.

Lösungsvorschlag

Iterativ heißt, dass der Entwikclungsprozess mehrfach wiederholt wird: statt den „Wasserfall“ einmal zu durchlaufen, werden „kleine Wasserfälle“ hintereinander gesetzt.

(b) Erklären Sie den Begriff inkrementelle Softwareentwicklung und grenzen Sie ihn von iterativer Softwareentwicklung ab.

Lösungsvorschlag

Bei der inkrementellen Entwicklung wird das System Schritt für Schritt fertig gestellt. D. h., dass ein Prototyp immer etwas mehr kann als der Prototyp davor. Dies wird durch die iterative Entwicklung unterstützt, da bei jeder Wiederholung des Entwicklungsprozesses ein neues Inkrement entsteht, d. h. ein neuer Prototyp, der mehr Funktionalitäten benutzt als der vorangegangene.

(c) Nennen Sie jeweils zwei Vor- und Nachteile eines iterativen und inkrementellen Vorgehens im Vergleich zum Wasserfallmodell.

Lösungsvorschlag

Vorteile:

- Risiken können früher erkannt werden.
- volatile Anforderungen können besser berücksichtigt werden.
- inkrementelle Auslieferung wird erleichtert.

Nachteile:

- komplexeres Projektmanagement
- schwerer messbar
- (Mehrarbeit)[a]

[a] Quelle: https://www.pst.ifi.lmu.de/Lehre/WS0607/pm/vorlesung/PM-02-Prozess.pdf

66116 / 2013 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 3

(a) Nennen Sie jeweils einen Vorteil und einen Nachteil für Qualitätssicherung durch *„Testing“* bzw. durch *„Model Checking“*.

Lösungsvorschlag

Qualitätssicherung durch „Testing“

Vorteil schnell, Massentests

Nachteil keine 100% Garantie, dass alles getestet ist

Qualitätssicherung durch „Model Checking“

Vorteil mathematischer Beweis

Nachteil teilweise langwierig / nicht möglich

(b) Definieren Sie den Begriff „Refactoring“.

Lösungsvorschlag

Verbesserung der Code-/Systemstruktur mit dem Ziel einer besseren Wartbarkeit

- Dokumentation, Namen, Kommentare
- keine neuen Funktionalitäten
- bessere Struktur, einheitlich, einfacher

(c) Begründen Sie, warum bei der Entwicklung nach der Methode des „eXtreme Programming“ langfristig gesehen Refactorings zwingend notwendig werden.

Lösungsvorschlag

Im „eXtreme Programming“ wird das Projekt kontinuierlich aufgebaut, somit ist ein Refactoring, auch aufgrund des Pair-Programmings, auf lange Sicht gesehen notwendig.

(d) Wie wird in der Praxis während und nach erfolgtem Refactoring sichergestellt, dass keine neuen Defekte eingeführt werden bzw. wurden?

Lösungsvorschlag

Re-testing → erneute Verifikation mit Tests nach Refactoring

(e) Worin besteht der Unterschied zwischen „White-Box-Testing“ und „Black-Box-Testing“?

Lösungsvorschlag

White-Box-Testing: Struktur-Test → Wie funktioniert der Code?

Black-Box-Testing: Funktionstest → Das nach außen sichtbare Verhalten wird getestet.

(f) Nennen Sie vier Qualitätsmerkmale von Software.

Lösungsvorschlag

Änderbarkeit, Wartbarkeit, gute Dokumentation, Effizienz, Funktionalität (→ Korrektheit), Zuverlässigkeit, Portabilität

(g) Worin besteht der Unterschied zwischen funktionalen und nicht-funktionalen Anforderungen?

Lösungsvorschlag

Funktionale Anforderung: Anforderung an die Funktionalität des Systems, also „Was kann es?“

Nicht-funktionale Anforderung: Design, Programmiersprache, Performanz

(h) Was verbirgt sich hinter dem Begriff „Continuous Integration” ?

Lösungsvorschlag

Continuous Integration: Das fertige Modul wird sofort in das bestehende Produkt integriert. Die Integration erfolgt also schrittweise und nicht erst, wenn alle Module fertig sind. Somit können auch neue Funktionalitäten sofort hinzugefügt werden (→ neue Programmversion).

(i) Nennen Sie sechs Herausstellungsmerkmale des *„eXtreme Programming“* Ansatzes.

Lösungsvorschlag

- Werte: Mut, Respekt, Einfachheit, Feedback, Kommunikation
- geringe Bedeutung von formalisiertem Vorgehen, Lösen der Programmieraufgabe im Vordergrund
- fortlaufende Iterationen
- Teamarbeit und Kommunikation (auch mit Kunden)
- Ziel: Software schneller und mit höherer Qualität bereitstellen, höhere Kundenzufriedenheit

- Continuous Integration und Testing, Prototyping
- Risikoanalysen zur Risikominimierung
- YAGNI (You ain't gonna need it) → nur die Features, die gefordert sind, umsetzen; kein Vielleicht braucht man's... "

(j) Was versteht man unter einem Unit-Test? Begründen Sie, warum es unzureichend ist, wenn eine Test-Suite ausschließlich Unit-Tests enthält.

Lösungsvorschlag

Unter einem Unit-Test versteht man den Test eines einzelnen Software-Moduls oder auch nur einer Methode. Dies ist allein nicht ausreichend, da man so nichts über das Zusammenspiel der Module aussagen kann.

(k) Nennen Sie jeweils eine Methodik, mit welcher in der Praxis die Prozesse der *„Validierung"* und der *„Verifikation"* durchgeführt werden.

Lösungsvorschlag

Methodik der Verifikation: Testen, wp-Kalkül, Model Checking

Methodik der Validierung: Kundentest, Kundengespräch (Spezifiaktion durchsprechen)

(l) Grenzen Sie die Begriffe *„Fault"* und *„Failure"* voneinander ab.

Lösungsvorschlag

Fault: interner Fehlerzustand, der nach außen nicht nicht sichtbar werden muss, aber kann.

Failure: Systemfehler / Fehlerzustand, der nach außen sichtbar wird.

66116 / 2016 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 1

(a) Welche Kriterien werden bei der Zielbestimmung im Pflichtenheft unterschieden?

Lösungsvorschlag

Im Rahmen der Zielbestimmung werden die Ziele des Produktes in einer Art Prioritätenliste exakt eingegrenzt, die in drei weitere Kategorien gegliedert ist, die als **Muss-**, **Wunsch-** und **Abgrenzungskriterien** bezeichnet werden.

Musskriterien Zu den Musskriterien zählen die Produktfunktionen, die für ein Funktionieren des Produktes unabdingbar sind. Ohne ihre Umsetzung geht einfach nichts, deshalb müssen sie erfüllt werden.

Wunschkriterien Die nächste Kategorie, die Wunschkriterien, sind zwar entbehrlich, wurden aber ausdrücklich vom Auftraggeber gefordert. Ihre Umsetzung im Produkt ist also genauso eine Pflicht, wie die der ersten Kategorie, das Produkt würde aber auch ohne ihre Implementierung seinen Betrieb korrekt ausführen.

Abgrenzungskriterien Die letzte Kategorie beschreiben die Abgrenzungskriterien. Sie sollen die Grenzen des Produktes klarmachen und sind von daher beinahe wichtiger als die beiden ersten Fälle denn sie hindern die Entwickler daran, über alle Ziele hinauszuschießen.

`https://st.inf.tu-dresden.de/SalesPoint/v3.2/tutorial/stepbystep2/pflh.html`

(b) Nennen Sie drei Einsatzziele von Softwaremaßen.

Lösungsvorschlag

Aus Baumann K. (1997) Softwaremaße. In: Unterstützung der objektorientierten Systemanalyse durch Softwaremaße. Beiträge zur Wirtschaftsinformatik, vol 23. Physica, Heidelberg. `https://doi.org/10.1007/978-3-662-13273-9_2`

Der Einsatz von Softwaremaßen kann vielfältige Vorteile mit sich bringen. Folgende Ziele können mit der Anwendung von Softwaremaßen verfolgt werden:

- Durch eine Bewertung kann überprüft werden, inwieweit einmal gestellte *Qualitätsanforderungen* erfüllt wurden.
- Eine Bewertung bereits erstellter oder noch zu erstellender Softwareprodukte kann Hilfestellung bei Entscheidungen, die für die *Projektplanung* oder das Entwicklungsmanagement zu treffen sind, leisten. So können Softwaremaße zur Einschätzung des notwendigen Aufwands, der zu erbringenden Kosten und des Personalbedarfs herangezogen werden.
- Ein möglichst frühzeitiges *Aufzeigen von Schwachstellen im Systemdesign*, die Beurteilung der Auswirkungen neuer Techniken oder Tools auf die Produktivität der Entwickler oder auf die Qualität des entwickelten Produkts sowie die Überprüfung der Einhaltung festgelegter Designstandards sind weitere mögliche Einsatzfelder für Softwaremaße.
- Darüber hinaus ist der Einsatz von Softwaremaßen *Teil umfassender Konzepte zum Softwarequalitätsmanagement*. Softwaremaße bzw. die zugehörige Meßergebnisse tragen zunächst dazu bei, die betrachteten Produkte oder Prozesse besser zu verstehen oder hinsichtlich interessierender Eigenschaften zu bewerten.

`https://link.springer.com/chapter/10.1007%2F978-3-662-13273-9_2`

(c) Welche Arten von Softwaremaßen werden unterschieden?

Lösungsvorschlag

Die Unterscheidung nach dem Messgegenstand ergibt folgende Arten von Maßen:

Externe Produktmaße. Diese beschreiben Merkmale des Produkts, die von

außen sichtbar sind, wenngleich eventuell nur indirekt. Dazu gehören insbesondere Maße für Qualitätsattribute wie die Wartbarkeit, die Zuverlässigkeit, die Benutzbarkeit, die Effizienz etc.

Interne Produktmaße. Basis für die Charakterisierung externer Produktattribute sind interne Attribute, die sich zumindest zum Teil besser messen lassen. Hierzu gehören z. B. die Größe, die Modularität und die Komplexität.

Prozessmaße. Diese charakterisieren Eigenschaften des Produktionsprozesses, z. B. die typische Produktivität, Kostenaufteilung auf bestimmte Arbeiten, Dauer von Arbeitsschritten etc.

Ressourcenmaße. Diese charakterisieren Eigenschaften der im Prozess verwendeten Ressourcen, z. B. die Auslastung von Rechnern, die typische Produktivität und Fehlerrate eines bestimmten Ingenieurs etc.

`http://page.mi.fu-berlin.de/prechelt/swt2/node5.html`

Was sind die 3 Arten der (einfachen) Maße zur Messung von Software?

https://quizlet.com/de/339799466/softwaretechnik-theorie-flash-cards/

- Größenorientiert/nach Größe
- Funktionsorientiert/nach Funktion
- Objektorientiert/nach Objekt

(d) Nennen Sie drei Merkmale evolutionärer Softwareentwicklung.

Lösungsvorschlag

Wie für jede Form der Software-Entwicklung stellt sich die Frage nach einem geeigneten methodischen Ansatz. Generell lässt sich der Prozess der Software-Entwicklung in die folgenden drei Abschnitte gliedern:

(i) von der Problembeschreibung bis zur Software-Spezifikation (auch Problemanalyse, Systemanalyse, Requirements Engineering bzw. frühe Phasen der Software-Entwicklung ge- nannt),

(ii) die Software-Entwicklung im engeren (technischen) Sinn,

(iii) die Integration der Software in den Anwendungszusammenhang.

Diese Einteilung gilt ebenso für die evolutionäre Software-Entwicklung, allerdings wird hier der Schwerpunkt anders gelegt. Steht beim klassischen Ansatz in der Regel die Software-Entwicklung im engeren Sinn im Mittelpunkt, so werden beim evolutionären Ansatz alle drei Abschnitte möglichst gleichgewichtig und im Zusammenhang betrachtet. Daraus resultiert eine natürliche Tendenz zu einer zyklischen Vorgehensweise.

Daneben gibt es weitere Querbezüge: Orientiert sich die Software-Entwicklung an einem festen Ziel, verfolgt dieses jedoch durch schrittweises Erweitern zu-

nächst unvollkommener Teillösungen, so wird dieses Vorgehen inkrementell genannt. Unfertige Versuchsanordnungen oder Teilsysteme, die am Beginn einer solchen Entwicklung stehen, werden oft als Prototypen bezeichnet. Wählt man Prototyping als Vorgehensweise, so kann entweder das spätere Produkt inkrementell aus einem oder mehreren Prototypen weiterentwickelt werden oder man setzt nach einer (Wegwerf-) Prototypphase neu auf. Beides sind Formen evolutionärer Entwicklung.

`http://waste.informatik.hu-berlin.de/~dahme/edidahm.pdf`

(e) Nennen Sie drei Vorteile des Einsatzes von Versionsverwaltungssoftware.

Lösungsvorschlag

(i) Möglichkeit, auf ältere Entwicklungsversionen zurückzukehren, wenn sich die aktuelle als nicht lauffähig bzw. unsicher erwiesen hat (z. B. git revert)

(ii) Möglichkeit, dass mehrere Entwickler gleichzeitig an ein- und denselben Softwareprojekt arbeiten und Möglichkeit, dass ihre Entwicklungen durch das Versionsverwaltungssystem zusammengeführt werden können (z. B. git merge)

(iii) Möglichkeit, den Zeitpunkt oder den Urheber eines Softwarefehlers ausfindig zumachen (z. B. git blame)

(f) Worin besteht der Unterschied zwischen funktionalen und nicht-funktionalen Anforderungen?

Lösungsvorschlag

Funktionale Anforderung: Anforderung an die Funktionalität des Systems, also „Was kann es?“

Nicht-funktionale Anforderung: Design, Programmiersprache, Performanz

(g) Was verbirgt sich hinter dem Begriff *„Continuous Integration“*?

Lösungsvorschlag

Continuous Integration: Das fertige Modul wird sofort in das bestehende Produkt integriert. Die Integration erfolgt also schrittweise und nicht erst, wenn alle Module fertig sind. Somit können auch neue Funktionalitäten sofort hinzugefügt werden (*rightarrow* neue Programmversion).

66116 / 2016 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 1

Ordnen Sie die folgenden Aussagen entsprechend ihres Wahrheitsgehaltes in einer Tabelle der folgenden Form an:

Kategorie	WAHR	FALSCH
X	X1, X3	X2
Y	Y2	Y1
...	...	...

A Allgemein

A1 Im Software Engineering geht es vor allem darum qualitativ hochwertige Software zu entwickeln.

A2 Software Engineering ist gleichbedeutend mit Programmieren.

B Vorgehensmodelle

B1 Die Erhebung und Analyse von Anforderungen sind nicht Teil des Software Engineerings.

B2 Agile Methoden eignen sich besonders gut für die Entwicklung komplexer und sicherer Systeme in verteilten Entwicklerteams.

B3 Das Spiralmodell ist ein Vorläufer sogenannter Agiler Methoden.

C Anforderungserhebung

C1 Bei der Anforderungserhebung dürfen in keinem Fall mehrere Erhebungstechniken (z. B. Workshops, Modellierung) angewendet werden, weil sonst Widersprüche in Anforderungen zu, Vorschein kommen könnten.

C2 Ein Szenario beinhaltet eine Menge von Anwendungsfällen.

C3 Nicht-funktionale Anforderungen sollten, wenn möglich, immer quantitativ spezifiziert werden.

D Architekturmuster

D1 Schichtenarchitekturen sind besonders für Anwendungen geeignet, in denen Performance eine wichtige Rolle spielt.

D2 Das Black Board Muster ist besonders für Anwendungen geeignet, in denen Performance eine wichtige Rolle spielt.

D3 „Dependency Injection“ bezeichnet das Konzept, welches Abhängigkeiten zur Laufzeit reglementiert.

E UML

E1 Sequenzdiagramme beschreiben Teile des Verhaltens eines Systems.

E2 Zustandsübergangsdiagramme beschreiben das Verhalten eines Systems.

E3 Komponentendiagramme beschreiben die Struktur eines Systems.

F Entwurfsmuster

F1 Das MVC Pattern verursacht eine starke Abhängigkeit zwischen Datenmodell und Benutzeroberfläche.

F2 Das Singleton Pattern stellt sicher, dass es zur Laufzeit von einer bestimmten Klasse höchstens ein Objekt gibt.

F3 Im Kommando Enwurfsmuster (engl. „Command Pattern“) werden Befehle in einem sog. Kommando-Objekt gekapselt, um sie bei Bedarf rückgängig zu machen.

G Testen

G1 Validation dient der Überprüfung von Laufzeitfehlern.

G2 Testen ermöglicht sicherzustellen, dass ein Programm absolut fehlerfrei ist.

G3 Verifikation dient der Überprüfung, ob ein System einer Spezifikation entspricht.

Lösungsvorschlag

Kategorie	WAHR	FALSCH
A	A1	A2
B	B3	B1, B2
C	C3	C1, C2
D	D3	D1, D2
E	E1, E2, E3	
F	F2, F3	F1
G	G3	G1[a], G2[b]

[a] Validierung: Prüfung der Eignung beziehungsweise der Wert einer Software bezogen auf ihren Einsatzzweck: „Wird das richtige Produkt entwickelt?“

[b] Ein Softwaretest prüft und bewertet Software auf Erfüllung der für ihren Einsatz definierten Anforderungen und misst ihre Qualität.

66116 / 2020 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 5

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

Lösungsvorschlag

(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.

Lösungsvorschlag

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.

Lösungsvorschlag

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?

Lösungsvorschlag

Wasserfallmodell
Prozessmodelle
Kontinuierliche Integration (Continuous Integration)
EXtreme Programming
Softwaremaße

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

66116 / 2020 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 1

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

Lösungsvorschlag

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.

Lösungsvorschlag

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.

Lösungsvorschlag

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.

Lösungsvorschlag SCRUM

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.

Lösungsvorschlag

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.

Lösungsvorschlag

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.

Lösungsvorschlag

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.

66116 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 2

(a) Erklären Sie den Unterschied zwischen *iterativen* und *inkrementellen* Entwicklungsprozessen. Nennen Sie zudem je ein Prozessmodell als Beispiel.

Lösungsvorschlag

Iterativ: Ein Entwicklungszyklus wird immer wieder durchlaufen:

Planung $\rightarrow$ Implementierung $\rightarrow$ Testung $\rightarrow$ Evaluation.

Mit jeder Iteration wird das Produkt verfeinert. Das Wasserfallmodell in seiner normalen Form würde dies nicht erfüllen, da hier alle Phasen nur einmal durchlaufen werden.

Beispiel: Agile Programmierung, erweitertes Wasserfallmodell

Inkrementell: Das Projekt wird in einzelne Teile zerlegt. An jedem Teilprojekt kann bestenfalls separat gearbeitet werden. In den einzelnen Teilprojekten kann dann ebenfalls wieder iterativ gearbeitet werden, die beiden Methoden schließen sich gegenseitig also nicht aus.

Beispiel: Agile Programmierung, V-Modell XT, Aufteilung in Teilprojekte, die alle mit Wasserfallmodell bearbeitet werden

(b) Nennen und erklären Sie kurz die vier Leitsätze der agilen Softwareentwicklung laut *Agilem Manifest*.

Lösungsvorschlag

(i) **Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge**
Ein festgelegter Prozess, der nicht sinnvoll von den beteiligten Personen umgesetzt werden kann, ist nicht sinnvoll und sollte nicht durchgeführt werden. Es geht darum, die Stärken der Mitarbeiter einzusetzen und sich nicht sklavisch an Abläufen zu orientieren. Es geht darum, Freiräume zu schaffen, um das Potential des Einzelnen voll entfalten zu lassen.

SCRUM
SCRUM

(ii) **Funktionierende Software ist wichtiger als umfassende Dokumentation**
Der Kunde ist in erster Linie an einem funktionierenden Produkt interessiert. Es soll möglichst früh ein Prototyp entstehen, der dann im weiteren Prozess an die Bedürfnisse des Kunden angepasst wird. Eine umfassende Dokumentation kann am Ende immer noch ergänzt werden.

(iii) **Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlung**
Anforderungen und Spezifikationen können sich flexibel ändern, dadurch kann der Kunde direkt Rückmeldung geben, ðes ist nicht wichtig im Vertrag kleinste Details zu regeln, sondern auf die Bedürfnisse des Kunden einzugehen. Daraus folgt direkt der letzte Punkt:

(iv) **Reagieren auf Veränderung ist wichtiger als das Befolgen eines Plans**
Softwareentwicklung ist ein dynamischer Prozess. Das starre Befolgen eines Plans widerspricht der Grundidee der agilen Programmierung.

(c) Beschreiben Sie die wesentlichen Aktivitäten in *Scrum* (agiles Entwicklungsmodell) inklusive deren zeitlichen Ablaufs. Gehen Sie dabei auch auf die *Artefakte* ein, die im Verlauf der Entwicklung erstellt werden.

Lösungsvorschlag

(i) **Initiale Projekterstellung:**
Festlegen eines Product Backlog durch den Product Owner, Erstellen von ersten User Stories. Diese entstehen gegebenenfalls durch den Kontakt mit den Stakeholdern, falls der Product Owner nicht selbst der Kunde ist.

(ii) **Sprint planning:**
Der nächste Sprint (zwischen 1 und 4 Wochen) wird geplant. Dabei wird ein Teil des Product Backlog als *Sprint Backlog* definiert und auf die einzelnen Entwickler verteilt. Es wird festgelegt, *was* implementiert werden soll (Product Owner anwesend) und *wie* das geschehen soll (Entwicklerteam). Während des Sprints findet jeden Tag ein *Daily Scrum* statt, ein fünfzehnminütiger Austausch zum aktuellen Stand. Am Ende des Sprints gibt es ein *Sprint Review* und das *Product Inkrement* wird evaluiert und das *Product Backlog* gegebenenfalls angepasst in Absprache mit *Product Owner* und *Stakeholdern*.

Artefakte:

- Product Backlog
- Sprint Backlog
- Product Increment

(d) Nennen und erklären Sie die *Rollen* in einem Scrum-Team.

Lösungsvorschlag

Product Owner Verantwortlich für das Projekt, für die Reihenfolge der Bearbeitung und Implementierung, ausgerichtet auf Maximierung und wirtschaftlichen Erfolg. Er führt und aktualisiert das *Product Backlog*.

Scrum Master Führungsperson, die das Umsetzen des Scrum an sich leitet und begleitet. Er mischt sich nicht mit konkreten Arbeitsanweisungen ein, sondern moderiert und versucht Hindernisse zu beseitigen, die einem Gelingen der Sprintziele im Weg sind.

Entwickler Sie entwicklen und implementieren die einzelnen Produktfunktionalitäten, die vom Product Owner festgelegt wurden, in eigenverantwortlicher Zeit. Sie müssen die Qualitätsstandards beachten.

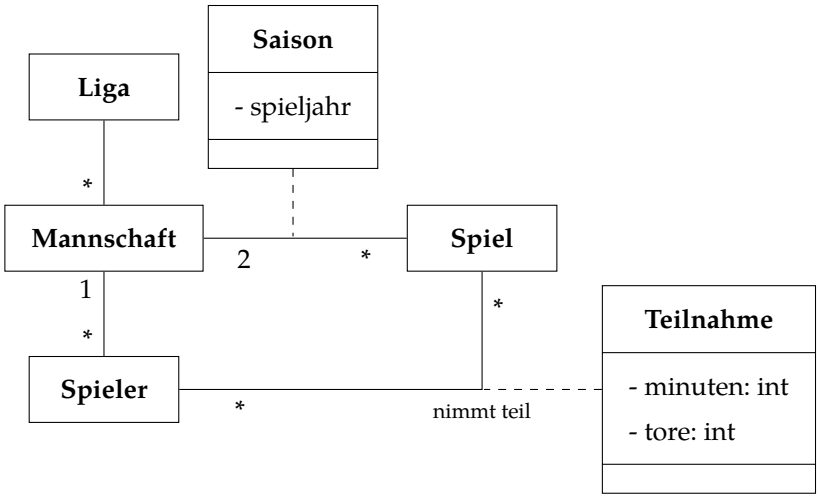
Modellierung

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

Modellieren Sie die folgende Situation als Klassendiagramm:

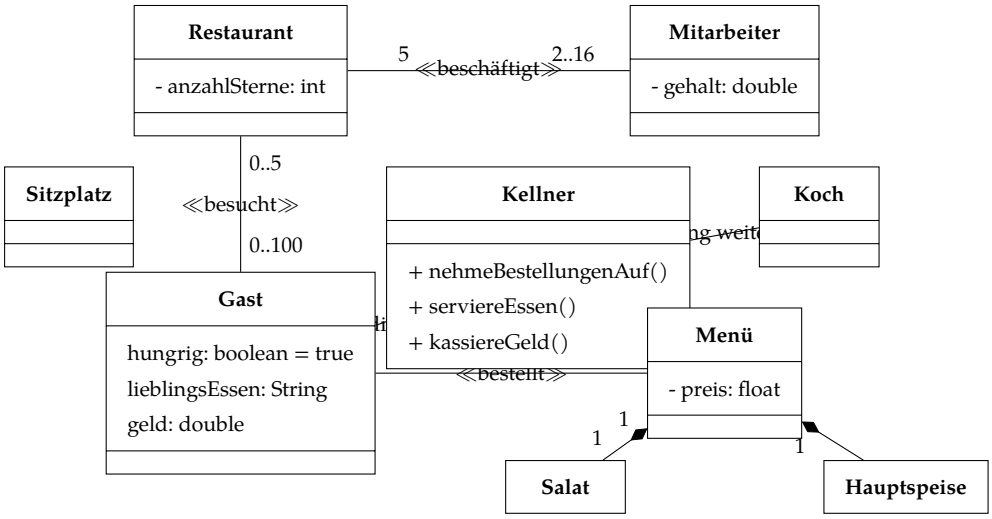
Fußballmannschaften einer Liga bestreiten während einer Meisterschaftsrunde Spiele gegen andere Mannschaften. Dabei werden in jeder Mannschaft Spieler für einen bestimmten Zeitraum (in Minuten) eingesetzt, die dabei eventuell Tore schießen.

Die Modellierung soll es ermöglichen, festzustellen, welcher Spieler in welchem Spiel wie lange auf dem Feld war und wie viele Tore geschossen hat. Ebenso soll es möglich sein, für jede Mannschaft festzustellen, gegen welche Mannschaft welche Ergebnisse erzielt wurden.



66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

In diesem kleinen Städtchen gibt es fünf Restaurants, aber egal, ob es sich um ein 2*-Restaurant, oder um ein 5*-Restaurant handelt, alle haben Gemeinsamkeiten. In jedem Restaurant sind mehrere **Mitarbeiter** angestellt. Im kleinsten Restaurant nur zwei, aber im größten verdienen 16 Mitarbeiter ihr Geld. Die Mitarbeiter bekommen in jedem Restaurant ein anderes *Gehalt*, aber alle machen ihre Arbeit. Die **Kellner** *nehmen bestellungen entgegen*, *servieren das Essen* und *kassieren das Geld* von den Gästen. Jeder Kellner gibt die Bestellungen der Gäste an die Köche weiter. Die **Köche** kennen verschiedene **Rezepte**, nach denen sie die **Menüs** dann zubereiten. Jedes Menü hat einen anderen *Preis*, aber es besteht nach Tradition immer aus einem **Salat** und einer **Hauptspeise**. An manchen Tagen ist kein **Gast** da. Dann werden Sie in ganz familiärem Rahmen bewirtet. Aber auch an guten Tagen gibt es, bei 100 Plätzen im größten Restaurant, sicher für jeden hungrigen Gast einen **Platz**. Wer in Gasthausen Essen geht sollte hungrig sein! Natürlich hat jeder Gast unterschiedlich viel Geld zur Verfügung, aber bei entsprechender Wahl des Restaurants reicht es sicher für sein Lieblingsessen.



66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

(a) Erstelle Sie ein Klassendiagramm zu folgender Aufgabenstellung:

Es gibt eine Klasse `Hund`, von der keine Objekte erzeugt werden sollen. Jeder Hund hat einen Namen (`String`), ein Alter in Jahren (`int`) und ein Gewicht in kg (`double`). Jeder Hund kann bellen, älter werden (um jeweils 1 Jahr), eine gewisse Menge an Hundefutter fressen und dabei schwerer werden, und Gassi gehen, wobei er wieder an Gewicht verliert. Hunde sind entweder Chihuahuas, Bernhardiner oder Schäferhunde. Chihuahuas bellen mit einem wuffwuff, wiegen durchschnittlich 2 kg, fressen maximal 0,09 kg pro Tag und verlieren ca. 0,04 kg pro Gassi-Gang. Ein Bernhardiner bellt mit einem lauten WAUWAU, wiegt durchschnittlich 95 kg, frisst maximal 0,5 kg pro Tag und verliert pro Gassi-Gang ca. 0,2 kg. Bernhardiner tragen eine Glocke um den Hals oder nicht. Ein Schäferhund bellt mit einem wauwau, wiegt durchschnittlich 30 kg, frisst maximal 0,3 kg pro Tag und verliert ca. 0,15 kg pro Gassi-Gang. Ein Schäferhund hat die Ausbildung zum Blindenhund absolviert oder nicht.

(b) Erstellen Sie nun zunächst die Klasse `Hund` mit den oben angegebenen Attributen und folgenden Methoden:

- Methode `bellen();`
- Methode `altern();`
- Methode `fressen(double futter);`
- Methode `fressen();` Verwendet als Futtermenge die maximale Menge pro Tag
- Methode `gassiGehen();`

(c) Erstellen Sie nun die Klassen `Chihuahua`, `Bernhardiner` und `Schaeferhund` mit ihren spezifischen Attributen und überschreiben Sie die in Teilaufgabe b) genannten Methoden.

(d) Erstellen Sie alle benötigten Getter- und Setter-Methoden.

Getter-Methode
Setter-Methode
Feld (Array)

(e) Erstellen Sie in einer Klasse `Zwinger` ein Hunde-Array `zwinger`, das Platz für zehn Hunde-Objekte hat, und folgende Methoden:

- Eine Methode `belegen()`, die die folgenden vier Hunde in die ersten vier Zwingerplätze setzt:
  - Chihuahua Tim, 2 Jahre alt, 1,8 kg schwer
  - Blindenhund Alex, 4 Jahre alt, 40 kg schwer
  - Berhardinerin Eva, 5 Jahre alt, 82 kg schwer
  - Schäferhündin Lilli, 3 Jahre alt, 34 kg schwer
- Eine Methode `fuettern()`, die alle Hunde im Zwinger mit ihrer maximalen Futtermenge versorgt.
- Eine Methode `fuetterzeit()`, die alle Hunde im Zwinger bellen lässt.
- Eine Methode `gassiGehen()`, die alle Hunde im Zwinger Gassi gehen lässt.

```
public abstract class Hund {
  protected String name;
  protected int alter;
  protected double gewicht;

  public Hund(String n, int a, double g) {
    name = n;
    alter = a;
    gewicht = g;
  }

  public abstract void bellen();

  public void altern() {
    alter = alter + 1;
  }

  public void fressen(double futter) {
    gewicht = gewicht + futter;
  }

  public abstract void fressen();

  public abstract void gassiGehen();

  public String getName() {
    return name;
  }

  public int getAlter() {
    return alter;
  }

  public double getGewicht() {
    return gewicht;
```

```
  }

  public void setName(String n) {
    name = n;
  }

  public void setAlter(int a) {
    if (a >= 0) {
      alter = a;
    }
  }

  public void setGewicht(double g) {
    if (g > 0) {
      gewicht = g;
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/hunde/Hund.java`

```
public class Bernhardiner extends Hund {
  private boolean glocke;

  public Bernhardiner(String n, int a, double g, boolean gl) {
    super(n, a, g);
    glocke = gl;
  }

  public void bellen() {
    System.out.println("WAUWAU");
  }

  public void fressen(double futter) {
    if (futter > 0 && futter < 0.5)
      super.fressen(futter);
  }

  public void fressen() {
    super.gewicht = super.gewicht + 0.5;
  }

  public void gassiGehen() {
    super.gewicht = super.gewicht - 0.2;
  }

  public boolean getGlocke() {
    return glocke;
  }

  public void setGlocke(boolean g) {
    glocke = g;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/hunde/Bernhardiner.java`

```java
public class Chihuahua extends Hund {
  public Chihuahua(String n, int a, double g) {
    super(n, a, g);
  }

  public void bellen() {
    System.out.println("wuffwuff");
  }

  public void fressen(double futter) {
    if (futter > 0 && futter < 0.09)
      super.fressen(futter);
  }

  public void fressen() {
    super.gewicht = super.gewicht + 0.09;
  }

  public void gassiGehen() {
    super.gewicht = super.gewicht - 0.04;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/hunde/Chihuahua.java`

```java
public class Schaeferhund extends Hund {
  private boolean blindenhund;

  public Schaeferhund(String n, int a, double g, boolean bl) {
    super(n, a, g);
    blindenhund = bl;
  }

  public void bellen() {
    System.out.println("wauwau");
  }

  public void fressen(double futter) {
    if (futter > 0 && futter < 0.3)
      super.fressen(futter);
  }

  public void fressen() {
    super.gewicht = super.gewicht + 0.3;
  }

  public void gassiGehen() {
    super.gewicht = super.gewicht - 0.15;
  }

  public boolean getBlind() {
    return blindenhund;
  }

  public void setBlind(boolean b) {
    blindenhund = b;
```

```
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/hunde/Schaeferhund.java`

```
public class Zwinger {
  // Attribute
  private Hund[] zwinger;

  // Konstruktor
  public Zwinger() {
    zwinger = new Hund[10];
  }

  // Methoden
  public void belegen() {
    zwinger[0] = new Chihuahua("Tim", 2, 1.8);
    zwinger[1] = new Schaeferhund("Alex", 4, 40.0, true);
    zwinger[2] = new Bernhardiner("Eva", 5, 82.0, false);
    zwinger[3] = new Schaeferhund("Lilli", 3, 34.0, false);
  }

  public void fuettern() {
    for (int i = 0; i < zwinger.length; i++) {
      if (zwinger[i] != null) {
        zwinger[i].fressen();
      }
    }
  }

  public void fuetterzeit() {
    for (int i = 0; i < zwinger.length; i++) {
      if (zwinger[i] != null) {
        zwinger[i].bellen();
      }
    }
  }

  public void gassiGehen() {
    for (int i = 0; i < zwinger.length; i++) {
      if (zwinger[i] != null) {
        zwinger[i].gassiGehen();
      }
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/hunde/Zwinger.java`

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

In der Kleintierpraxis Dr. Huf werden *Hunde*, *Katzen* und *Heimtiere*[1] behandelt. In der Praxissoftware werden von jedem Tier *Name*, *Alter* (in Jahren) und *Gewicht* (in kg mit

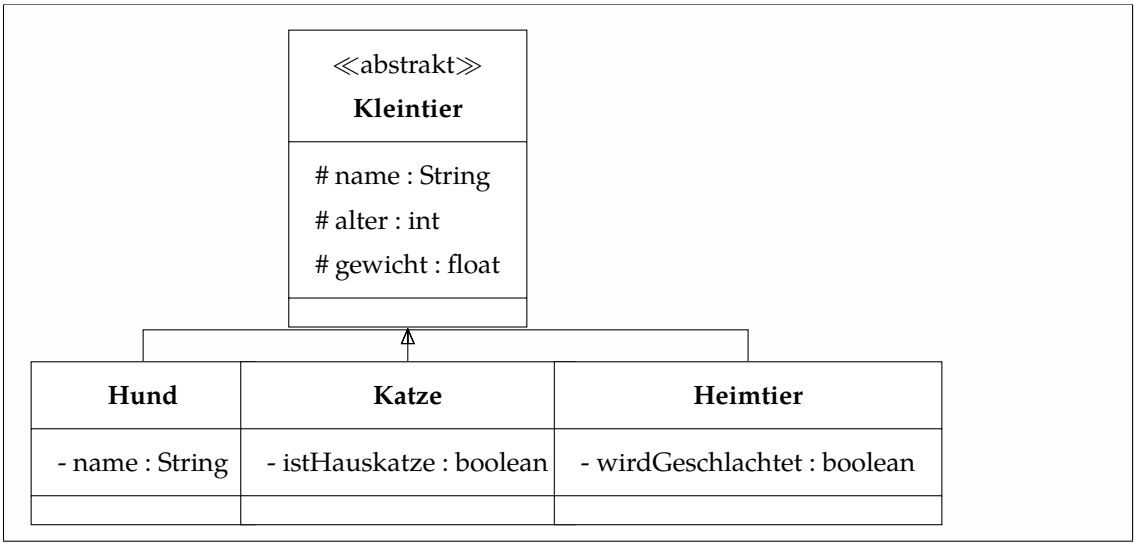
[1] Unter Heimtiere versteht man hier Kleintiere (d. h. keine Rinder, Pferde etc.), die zu Hause gehalten werden und keine Hunde oder Katzen sind (beispielsweise Meerschweinchen oder Kaninchen). Heim-

Klassendiagramm
Vererbung

mindestens 2 Dezimalen) erfasst. Bei einem *Hund* wird zusätzlich aufgenommen, ob er *reinrassig* ist, bei einer *Katze*, ob sie ausschließlich *in der Wohnung gehalten* wird, und beim *Heimtier*, ob es zur *Lebensmittellieferung* dient.

(a) Erstellen Sie ein entsprechendes Klassendiagramm (zunächst ohne Methoden), das den oben beschriebenen Sachverhalt geeignet erfasst.

Lösungsvorschlag



(b) Implementieren Sie die Klassen gemäß des Modells aus Teilaufgabe a in der Programmierumgebung BlueJ. Beachten Sie dabei die abstrakte Oberklasse!

Lösungsvorschlag

Klasse *Kleintier*

```
public abstract class Kleintier {
  protected String name;
  protected int alter;
  protected float gewicht;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Kleintier.java`

```
  public Kleintier(String name, int alter, float gewicht) {
    this.name = name;
    this.alter = alter;
    this.gewicht = gewicht;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Kleintier.java`

Klasse *Hund*

tiere können auch Nutztiere sein, die zur Schlachtung gehalten werden (z. B. Hühner). In diesem Fall muss ein Tierarzt darauf achten, nur Medikamente anzuwenden, die für Lebensmittel liefernde Tiere zugelassen sind.

```java
public class Hund extends Kleintier {
  private boolean reinrassig;

  public Hund(String name, int alter, float gewicht, boolean reinrassig) {
    super(name, alter, gewicht);
    this.reinrassig = reinrassig;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Hund.java`

Klasse *Katze*

```java
public class Katze extends Kleintier {
  @SuppressWarnings("unused")
  private boolean istHauskatze;

  public Katze(String name, int alter, float gewicht, boolean istHauskatze) {
    super(name, alter, gewicht);
    this.istHauskatze = istHauskatze;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Katze.java`

Klasse *Heimtier*

```java
public class Heimtier extends Kleintier {
  @SuppressWarnings("unused")
  private boolean wirdGeschlachtet;

  public Heimtier(String name, int alter, float gewicht, boolean
↪   wirdGeschlachtet) {
    super(name, alter, gewicht);
    this.wirdGeschlachtet = wirdGeschlachtet;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Heimtier.java`

(c) Die Praxissoftware verfügt über eine Methode `datenAusgeben()`, die den Namen und das Alter eines Tieres auf dem Bildschirm ausgibt. Fügen Sie im Klassendiagramm die Methode an geeigneter Stelle ein und implementieren Sie sie.

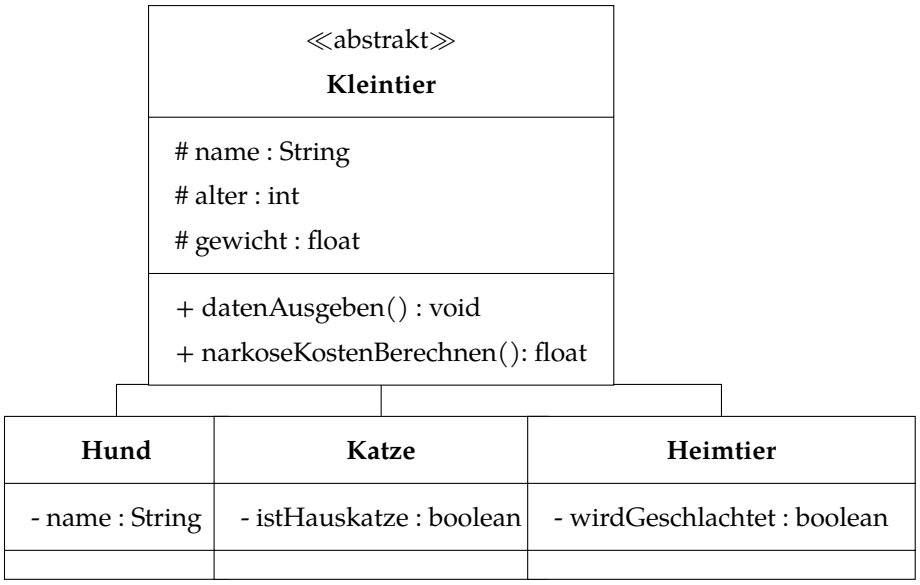
Lösungsvorschlag

Einfügen in der abstrakten Klasse *Heimtier*. Fertiges Klassendiagramm siehe 1 (d)

```java
    public void datenAusgeben()
    {
        System.out.println("Name: " + name + ", Alter: " + alter);
    }
```

(d) Die Kosten für eine Narkose setzen sich zusammen aus einer Grundgebühr, die von der Tierart abhängt, und aus den Kosten für das verwendete Narkotikum.

Diese wiederum berechnen sich aus dem Preis des Narkotikums pro kg Körpergewicht multipliziert mit dem Gewicht des Tieres. Ergänzen Sie das Klassendiagramm entsprechend um die Methode `narkosekostenBerechnen()`, die die Kosten für eine Narkose auf dem Bildschirm ausgibt, und implementieren Sie sie.



Lösungsvorschlag

Einfügen in der abstrakten Klasse *Heimtier*.

```
public class Hund extends Kleintier {
  private boolean reinrassig;

  public Hund(String name, int alter, float gewicht, boolean reinrassig) {
    super(name, alter, gewicht);
    this.reinrassig = reinrassig;
    narkoseGrundGebuehr = 1.50f;
  }

  public void datenAusgeben() {
    System.out.println("Name: " + name + ", Alter: " + alter + " Jahre");

    if (reinrassig) {
      System.out.println("Der Hund ist reinrassig.");
    } else {
      System.out.println("Der Hund ist nicht reinrassig.");
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Hund.java`

Einfügen in den Konstruktor der Klasse *Hund*:

```
narkoseGrundGebuehr = 1.50f;
```

Einfügen in den Konstruktor der Klasse *Katze*:

Klassendiagramm

```
narkoseGrundGebuehr = 1f;
```

Einfügen in den Konstruktor der Klasse *Heimtier*:

```
narkoseGrundGebuehr = 2f;
```

(e) Falls es sich bei dem zu behandelnden Tier um einen Hund handelt, soll die Methode `datenAusgeben()` (siehe Teilaufgabe c) dies zusammen mit dem Namen und dem Alter des Tieres auf dem Bildschirm ausgeben. Außerdem soll in diesem Fall ausgegeben werden, ob der Hund reinrassig ist oder nicht.

Lösungsvorschlag

Einfügen in der Klasse *Hund*:

```java
public void datenAusgeben() {
  System.out.println("Name: " + name + ", Alter: " + alter + " Jahre");

  if (reinrassig) {
    System.out.println("Der Hund ist reinrassig.");
  } else {
    System.out.println("Der Hund ist nicht reinrassig.");
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Hund.java`

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

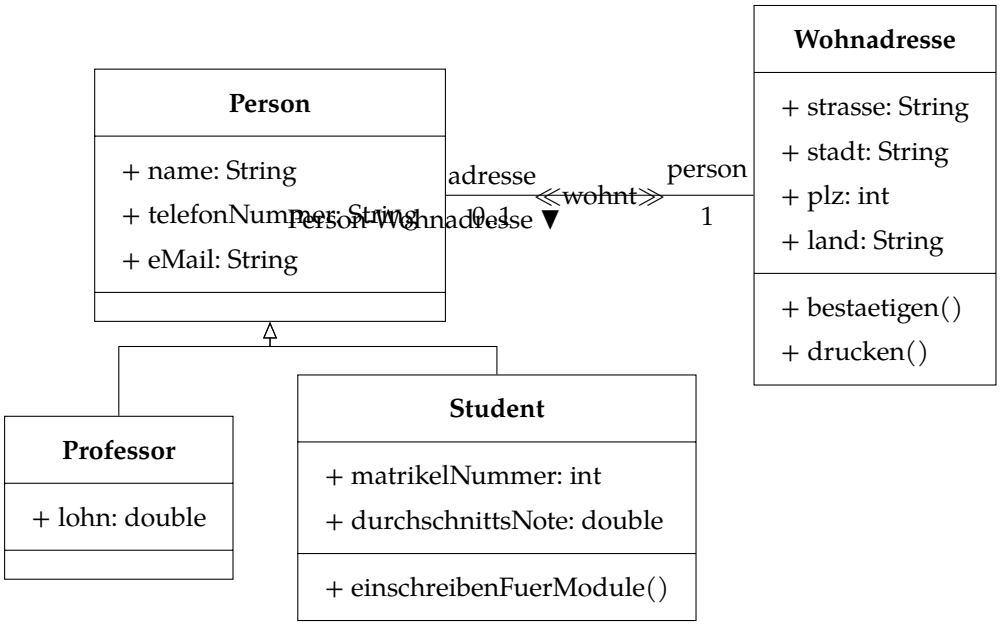
Gegeben ist der folgende Sachverhalt.

Jede **Person** hat einen *Namen*, eine *Telefonnummer* und *E-Mail*.

Jede **Wohnadresse** wird von nur einer Person bewohnt. Es kann aber sein, dass einige Wohnadressen nichtbewohnt sind. Den Wohnadressen sind je eine *Strasse*, eine *Stadt*, eine *PLZ* und ein *Land* zugeteilt. Alle Wohnadressen können bestätigt werden und als Beschriftung (für Postversand) gedruckt werden.

Es gibt zwei Sorten von **Personen**: **Student**, welcher sich für ein *Modul einschreiben* kann und **Professor**, welcher einen *Lohn* hat. Der Student besitzt eine *Matrikelnummer* und eine *Durchschnittsnote*.

Modellieren Sie diesen Sachverhalt mit einem UML Klassendiagramm.



66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

Max und Sarah haben ein Ziel. Jeder Gast, der ihr Restaurant, das „fleißige Bienchen“ besucht, soll so schnell wie möglich satt werden. Um das zu erreichen arbeiten sie auf Hochtouren. Wenn ein Gast sein Essen bestellt, wird das sofort in der Küche gehört. Max fängt direkt an das Gericht zuzubereiten. Man hat es noch nicht bewiesen, aber wahrscheinlich ist er der schnellste Koch der Welt und dementsprechend dauert das Kochen nur wenige Minuten. Ist das Gericht fertig rennt Sarah zu den Gästen und serviert es. Manchmal verliert sie dabei, weil sie so schnell rennt, Teile des Essens. Dann muss sie zurück in die Küche, in der das Gericht neu zubereitet wird. Wenn aber alles gut gegangen ist, kann der Gast schnell essen. Gäste mit kleinem Hunger sind nach dem Essen satt. Hungrige Gäste bestellen noch einen Nachtisch der schon bereit steht, damit keine Zeit verloren geht. Ist dieser aufgegessen sind auch die hungrigsten Gäste satt. Sarah und Max haben herausgefunden, dass die Gäste beim Bezahlen der Rechnung viel mehr Trinkgeld geben, wenn sie innerhalb von einer halben Stunde satt sind. Aber egal, wie viel Geld sie bekommen. Max und Sarah sind glücklich über jeden Gast der sie besucht.

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

Ordnen Sie die gegebenen Diagrammarten der Fragestellung zu, die das jeweilige Diagramm am besten beantwortet.

(a) Wie sind die Daten und das Verhalten meines Systems im Detail strukturiert?

Lösungsvorschlag

Klassendiagramm

(b) Wie sieht ein Schnappschuss meines Systems zur Ausführungszeit aus?

Lösungsvorschlag

Vererbung
Generalisierung
Spezialisierung

Objektdiagramm

(c) Wie verhält sich das System in einem bestimmten Zustand bei gewissen Ereignissen?

Lösungsvorschlag

Zustandsdiagramm

(d) Wie läuft die Kommunikation in meinem System ab?

Lösungsvorschlag

Sequenzdiagramm

(e) Wie realisiert mein System ein bestimmtes Verhalten?

Lösungsvorschlag

Aktivitätsdiagramm

(f) Was soll mein geplantes System leisten?

Lösungsvorschlag

Anwendungsfalldiagramm

(g) Welche Teile einer komplexen Struktur arbeiten wie zusammen, um eine bestimmte Funktion zu erfüllen?

Lösungsvorschlag

Kommunikationsdiagramm

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

Eine kleine Bank bietet drei Arten von Konten an: Girokonten, Sparkonten und Geschäftskonten. Alle drei Kontoarten haben die Methoden `Einzahlen`, `Abheben` und `KontostandGeben` sowie die Attribute `kontostand` und `kontonummer`.

- Sparkonten haben einen Zinssatz und eine Methode `Verzinsen`, die den Jahreszins zum Guthaben addiert. Maximalbetrag beim Abheben ist der aktuelle Kontostand.
- Girokonten können um bis 2000 € überzogen werden (Dispokredit).
- Geschäftskonten haben einen variablen Dispokredit, der über die Methode `DispoKreditSetzen` festgelegt wird; der Startwert für den Dispokredit wird mit dem Konstruktor beim Einrichten des Kontos als Parameter mitgegeben.

(a) Überlege dir, welche Konten Generalisierungen bzw. Spezialisierungen anderer Konten sind. Warum ist es sinnvoll, eine Klasse Konto als oberste Klasse Generalisierungshierarchie einzuführen?

Lösungsvorschlag Klassendiagramm

Da alle Konten die Methoden `einzahlen()`, `abheben()` und `kontostandGeben()` sowie die Attribute `kontostand` und `kontonummer` besitzen, bietet sich deren Verwaltung in einer einzigen Klasse an. Da jede Kontoart aber auch individuelle Eigenschaften bzw. Methoden hat, muss es für jede auch eine eigene Klasse geben. Daher bietet es sich an, die Gemeinsamkeiten in eine Oberklasse `Konto` auszulagern (Generalisierung).

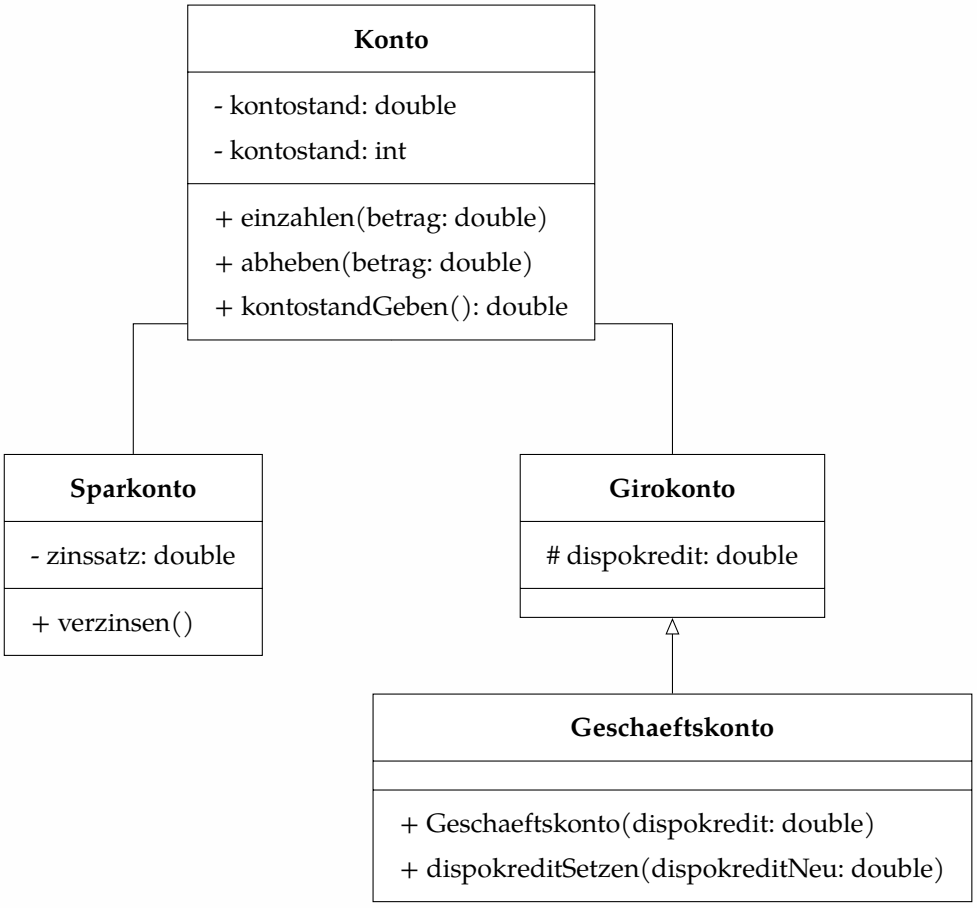
(b) Entwirf ein Klassendiagramm für die Klassen `Konto`, `Girokonto`, `Sparkonto`, `Geschaeftskonto`.

Lösungsvorschlag

Da das Geschäftskonto genauso wie das Girokonto einen Dispokredit besitzt, dieser nur anders festgelegt wird, wurde bei der Modellierung die Klasse `Geschaeftskonto` als Unterklasse der Klasse `Girokonto` umgesetzt.

Die Oberklasse `Konto` wurde abstrakt modelliert, da von ihr direkt keine Objekte erzeugt werden können.

Die Attribute `kontostand` und `kontonummer` in der Klasse `Konto` haben den Sichtbarkeitsmodifikator `private`, da die Unterklassen nie direkt auf die Attribute zugreifen, sondern die zur Verfügung stehenden Methoden dafür verwenden.



(c) Implementiere die Klassen in einem eigenen Projekt und teste die vorhandenen Methoden.

Lösungsvorschlag

```java
public abstract class Konto {
  private double kontostand;
  @SuppressWarnings("unused")
  private int kontonummer;

  public Konto(int kontonummer) {
    this.kontonummer = kontonummer;
    kontostand = 0;
  }

  public void einzahlen(double betrag) {
    kontostand = kontostand + betrag;
  }

  public void abheben(double betrag) {
    // Ob abgehoben werden darf, entscheidet die Methode der jeweiligen
 ↪  Unterklasse.
    kontostand = kontostand - betrag;
  }

  public double kontostandGeben() {
    return kontostand;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/pu_3/bankverwaltung/Konto.java`

```java
public class Sparkonto extends Konto {
  private double zinssatz;

  public Sparkonto(int kontonummer, double zinssatz) {
    super(kontonummer);
    // Aufruf des Konstruktors der Oberklasse
    this.zinssatz = zinssatz;
  }

  /**
   * Überschreiben der Methode abheben() aus der Oberklasse. Ist genügend
 ↪  Geld auf
   * dem Sparkonto...? dann darf man den gewünschten Betrag abheben, sonst
 ↪  nicht.
   */
  public void abheben(double betrag) {
    if (kontostandGeben() >= betrag) {
      super.abheben(betrag);
    }
  }

  /**
   * Der aktuelle Kontostand wird mit dem Zinssatz verrechnet und der sich
 ↪  daraus
```

```
    * ergebende Betrag (= Zinsen) dem Konto gutgeschrieben.
    */
  public void verzinsen() {
    einzahlen(kontostandGeben() * zinssatz);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/pu_3/bankverwaltung/Sparkonto.java`

```
public class Girokonto extends Konto {
  /**
   * Die Unterklasse muss auch auf das Attribut zugreifen können.
   */
  protected double dispokredit;

  public Girokonto(int kontonummer) {
    super(kontonummer);
    dispokredit = 2000;
  }

  /**
   * Die Methode abheben() kann direkt von der Oberklasse GIROKONTO und die
   * Methoden einzahlen() und kontostandGeben() von der Oberklasse KONTO
↪  genutzt
   * werden.
   *
   * Überschreiben der Methode abheben() der Klasse KONTO
   * Ist genügend Geld auf dem Konto bzw. reicht der Dispokredit aus...
   *  ...dann darf man den gewünschten Betrag abheben, sonst nicht.
   */
  public void abheben(double betrag) {//
    if (kontostandGeben() + dispokredit >= betrag) {
      super.abheben(betrag);
    }
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/pu_3/bankverwaltung/Girokonto.java`

```
public class Geschaeftskonto extends Girokonto {

  /**
   * Für das Geschäftskonto wird der Dispo individuell festgelegt.
   *
   * @param kontonummer Die Kontonummer
   * @param dispo Der maximale Rahme des Dispokredits.
   */
  public Geschaeftskonto(int kontonummer, double dispo) {
    super(kontonummer);
    dispokredit = dispo;
  }

  public void dispokreditSetzen(double dispokreditNeu) {
```

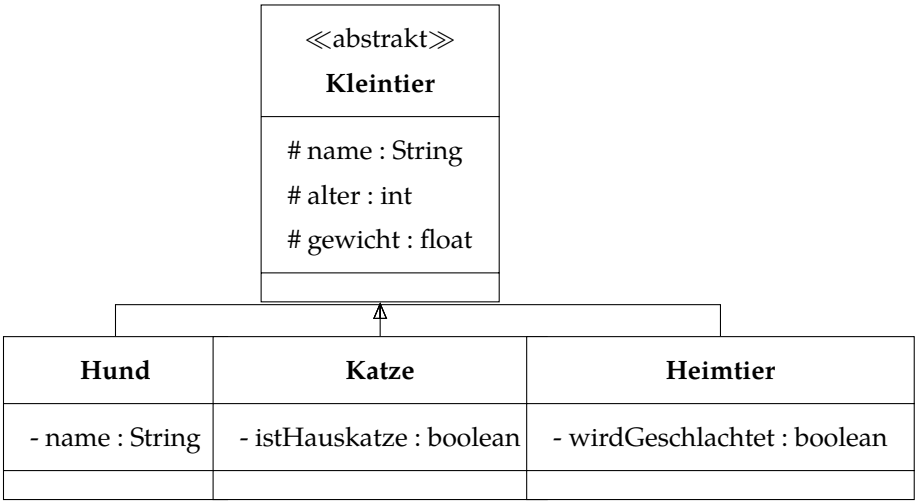
```
    dispokredit = dispokreditNeu;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/pu_3/bankverwaltung/Geschaeftskonto.java`

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

In der Kleintierpraxis Dr. Huf werden *Hunde*, *Katzen* und *Heimtiere*[2] behandelt. In der Praxissoftware werden von jedem Tier *Name*, *Alter* (in Jahren) und *Gewicht* (in kg mit mindestens 2 Dezimalen) erfasst. Bei einem *Hund* wird zusätzlich aufgenommen, ob er *reinrassig* ist, bei einer *Katze*, ob sie ausschließlich *in der Wohnung gehalten* wird, und beim *Heimtier*, ob es zur *Lebensmittellieferung* dient.

(a) Erstellen Sie ein entsprechendes Klassendiagramm (zunächst ohne Methoden), das den oben beschriebenen Sachverhalt geeignet erfasst.



(b) Implementieren Sie die Klassen gemäß des Modells aus Teilaufgabe a in der Programmierumgebung BlueJ. Beachten Sie dabei die abstrakte Oberklasse!

Klasse *Kleintier*

```
public abstract class Kleintier {
  protected String name;
  protected int alter;
  protected float gewicht;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Kleintier.java`

[2] Unter Heimtiere versteht man hier Kleintiere (d. h. keine Rinder, Pferde etc.), die zu Hause gehalten werden und keine Hunde oder Katzen sind (beispielsweise Meerschweinchen oder Kaninchen). Heimtiere können auch Nutztiere sein, die zur Schlachtung gehalten werden (z. B. Hühner). In diesem Fall muss ein Tierarzt darauf achten, nur Medikamente anzuwenden, die für Lebensmittel liefernde Tiere zugelassen sind.

```
  public Kleintier(String name, int alter, float gewicht) {
    this.name = name;
    this.alter = alter;
    this.gewicht = gewicht;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Kleintier.java`

Klasse *Hund*

```
public class Hund extends Kleintier {
  private boolean reinrassig;

  public Hund(String name, int alter, float gewicht, boolean reinrassig) {
    super(name, alter, gewicht);
    this.reinrassig = reinrassig;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Hund.java`

Klasse *Katze*

```
public class Katze extends Kleintier {
  @SuppressWarnings("unused")
  private boolean istHauskatze;

  public Katze(String name, int alter, float gewicht, boolean istHauskatze) {
    super(name, alter, gewicht);
    this.istHauskatze = istHauskatze;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Katze.java`

Klasse *Heimtier*

```
public class Heimtier extends Kleintier {
  @SuppressWarnings("unused")
  private boolean wirdGeschlachtet;

  public Heimtier(String name, int alter, float gewicht, boolean
↪   wirdGeschlachtet) {
    super(name, alter, gewicht);
    this.wirdGeschlachtet = wirdGeschlachtet;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Heimtier.java`

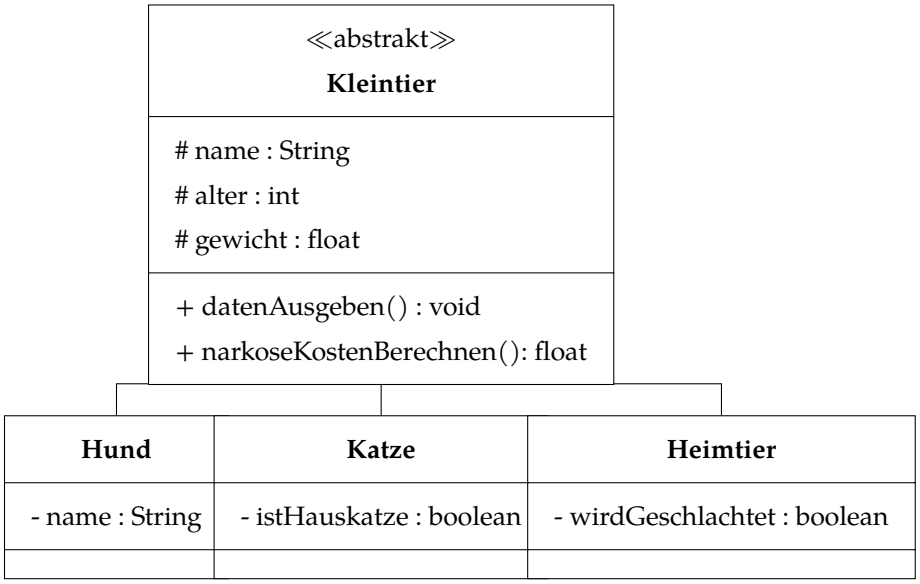
(c) Die Praxissoftware verfügt über eine Methode `datenAusgeben()`, die den Namen und das Alter eines Tieres auf dem Bildschirm ausgibt. Fügen Sie im Klassendiagramm die Methode an geeigneter Stelle ein und implementieren Sie sie.

Lösungsvorschlag

Einfügen in der abstrakten Klasse *Heimtier*. Fertiges Klassendiagramm siehe 1 (d)

```
public void datenAusgeben()
{
    System.out.println("Name: " + name + ", Alter: " + alter);
}
```

(d) Die Kosten für eine Narkose setzen sich zusammen aus einer Grundgebühr, die von der Tierart abhängt, und aus den Kosten für das verwendete Narkotikum. Diese wiederum berechnen sich aus dem Preis des Narkotikums pro kg Körpergewicht multipliziert mit dem Gewicht des Tieres. Ergänzen Sie das Klassendiagramm entsprechend um die Methode `narkosekostenBerechnen()`, die die Kosten für eine Narkose auf dem Bildschirm ausgibt, und implementieren Sie sie.



Lösungsvorschlag

Einfügen in der abstrakten Klasse *Heimtier*.

```
public class Hund extends Kleintier {
  private boolean reinrassig;

  public Hund(String name, int alter, float gewicht, boolean reinrassig) {
    super(name, alter, gewicht);
    this.reinrassig = reinrassig;
    narkoseGrundGebuehr = 1.50f;
  }

  public void datenAusgeben() {
    System.out.println("Name: " + name + ", Alter: " + alter + " Jahre");

    if (reinrassig) {
      System.out.println("Der Hund ist reinrassig.");
    } else {
```

```
      System.out.println("Der Hund ist nicht reinrassig.");
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Hund.java`

Einfügen in den Konstruktor der Klasse *Hund*:

```
narkoseGrundGebuehr = 1.50f;
```

Einfügen in den Konstruktor der Klasse *Katze*:

```
narkoseGrundGebuehr = 1f;
```

Einfügen in den Konstruktor der Klasse *Heimtier*:

```
narkoseGrundGebuehr = 2f;
```

(e) Falls es sich bei dem zu behandelnden Tier um einen Hund handelt, soll die Methode `datenAusgeben()` (siehe Teilaufgabe c) dies zusammen mit dem Namen und dem Alter des Tieres auf dem Bildschirm ausgeben. Außerdem soll in diesem Fall ausgegeben werden, ob der Hund reinrassig ist oder nicht.

Lösungsvorschlag

Einfügen in der Klasse *Hund*:

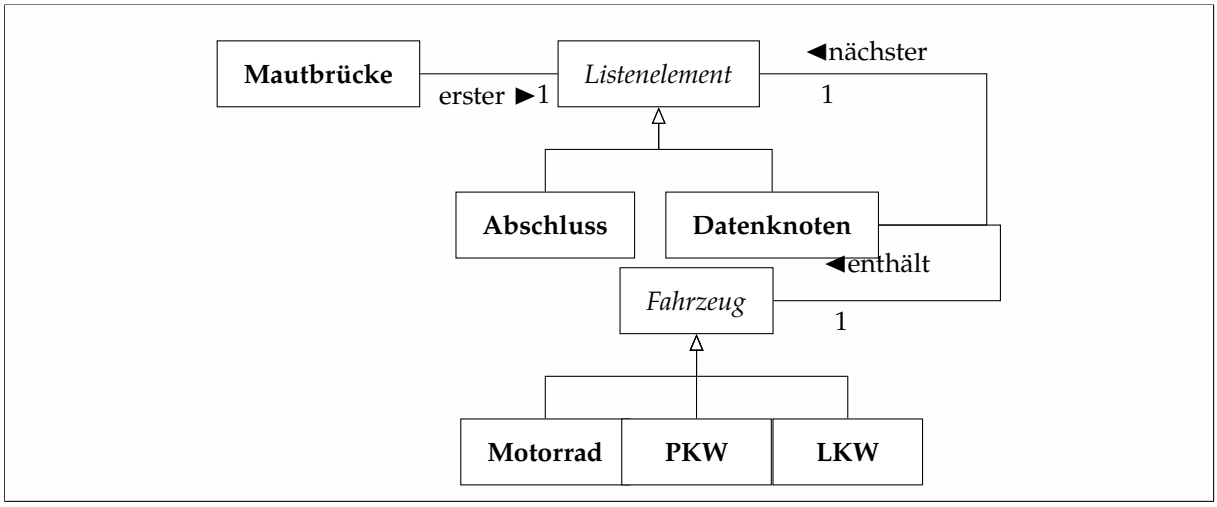
```
  public void datenAusgeben() {
    System.out.println("Name: " + name + ", Alter: " + alter + " Jahre");

    if (reinrassig) {
      System.out.println("Der Hund ist reinrassig.");
    } else {
      System.out.println("Der Hund ist nicht reinrassig.");
    }
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/oomup/klassendiagramm/kleintierpraxis/Hund.java`

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

Für die Umsetzung der Maut auf deutschen Autobahnen soll eine Java-basierte Lösung entworfen werden. Dazu sollen alle Fahrzeuge, die von einer Mautbrücken erfasst werden, in einer *einfach verketteten Liste* abgelegt werden. Um einen besseren Überblick über die Einnahmen zu erhalten, soll zwischen *LKWs*, *PKWs* und *Motorrädern* unterschieden werden. Als Informatiker schlagen Sie eine *heterogene Liste* zur Realisierung vor. Notieren Sie unter Verwendung des *Entwurfsmusters Kompositum* ein entsprechendes *Klassendiagramm* zur Realisierung der Lösung für eine Mautbrücke. Auf die Angabe von Attributen und Methoden kann verzichtet werden. Kennzeichen Sie in Ihrem Klassendiagramm die *abstrakten Klassen* und benennen Sie die bestehenden *Beziehungen*.

Lösungsvorschlag



66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

Erstellen Sie ein Deutsch-Englisch Wörterbuch. Verwenden Sie dazu eine einfach verkettete Liste mit Kompositum. Identifizieren Sie die benötigten Klassen, legen Sie das Wörterbuch an und implementieren Sie anschließend die geforderten Methoden.

- Ein Listenelement, welches immer jeweils auf seinen Nachfolger verweisen kann, enthält jeweils einen Eintrag des Wörterbuchs. Ein Eintrag besteht aus dem deutschen und dem zugehörigen englischen Wort. Diese können natürlich jeweils zurückgegeben werden.

- Mit der Methode `einfuegen (String deutsch, String englisch)` soll ein neuer Eintrag in das Wörterbuch eingefügt werden können. Wie in jedem Wörterbuch müssen die (deutschen) Einträge jedoch alphabetisch sortiert sein, sodass nicht an einer beliebigen Stelle eingefügt werden kann. Um die korrekte Einfügeposition zu finden, ist das Vergleichen von Strings notwendig. Recherchieren Sie dazu, wie die Methode `compareTo()` in Java funktioniert!

- Der Aufruf der Methode `uebersetze(String deutsch)` auf der Liste soll nun für ein übergebenes deutsches Wort die englische Übersetzung ausgeben.

Lösungsvorschlag

Klasse `WörterbuchEintrag`

Die abstrakte Klasse im Kompositumentwurfsmuster von der sowohl die primitive Klasse als auch die Behälterklasse erben.

```
public abstract class WoerterbuchEintrag {
  protected WortPaar nächstes;

  protected WortPaar gibNächstes () {
    return nächstes;
  }
```

```
  protected void setzeNächstes (WortPaar wortPaar) {
    nächstes = wortPaar;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/WoerterbuchEintrag.java`

Klasse `WortPaar`

Das Listenelement (die primitive Klasse im Kompositumentwurfsmuster).

```
public class WortPaar extends WoerterbuchEintrag {
  private final String deutsch;

  private final String englisch;

  public WortPaar(String deutsch, String englisch) {
    this.deutsch = deutsch;
    this.englisch = englisch;
  }

  public String gibDeutschesWort() {
    return deutsch;
  }

  public String gibEnglischesWort() {
    return englisch;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/WortPaar.java`

Klasse `Wörterbuch`

Die Behälterklasse im Kompositumentwurfsmuster.

```
 */
public class Woerterbuch extends WoerterbuchEintrag {

  public void einfügen(String deutsch, String englisch) {
    WortPaar wort = new WortPaar(deutsch, englisch);

    // Spezialbehandlung, wenn vor das erste Wortpaar des Wörterbuchs eingefügt
    // werden muss.
    WortPaar kopf = gibNächstes();
    if (kopf == null || kopf.gibDeutschesWort().compareTo(wort.gibDeutschesWort())
↪   >= 0) {
      wort.setzeNächstes(kopf);
      setzeNächstes(wort);
      return;
    }
    WortPaar vergleichsWort = gibNächstes();
```

```
    while (vergleichsWort.gibNächstes() != null
        &&
↪   vergleichsWort.gibNächstes().gibDeutschesWort().compareTo(wort.gibDeutschesWort())
↪   < 0) {
      vergleichsWort = vergleichsWort.gibNächstes();
    }
    wort.setzeNächstes(vergleichsWort.gibNächstes());
    vergleichsWort.setzeNächstes(wort);
  }

  public String übersetze(String deutsch) {
    if (gibNächstes() == null) {
      return "Noch keine Wörter im Wörterbuch.";
    }
    WortPaar wort = gibNächstes();
    while (wort != null) {
      if (wort.gibDeutschesWort().equals(deutsch)) {
        return wort.gibEnglischesWort();
      }
      wort = wort.gibNächstes();
    }
    return "Es konnte keine passende Übersetzung gefunden werden";
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/Woerterbuch.java`

Test

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class WoerterbuchTest {

  @Test
  public void methodeÜbersetze() {
    Woerterbuch wörterbuch = new Woerterbuch();
    assertEquals("Noch keine Wörter im Wörterbuch.",
↪   wörterbuch.übersetze("Wassermelone"));
  }


  @Test
  public void methodeEinfügen() {
    Woerterbuch wörterbuch = new Woerterbuch();
    wörterbuch.einfügen("Wassermelone", "Watermelon");
    assertEquals("Watermelon", wörterbuch.übersetze("Wassermelone"));
  }

  @Test
  public void sortierung() {
    Woerterbuch wörterbuch = new Woerterbuch();
    wörterbuch.einfügen("Wassermelone", "Watermelon");
```

```
    wörterbuch.einfügen("Apfel", "Apple");
    wörterbuch.einfügen("Zitrone", "Lemon");
    wörterbuch.einfügen("Birne", "Pear");
    wörterbuch.einfügen("Klementine", "Clementine");

    WortPaar paar;
    paar = wörterbuch.gibNächstes();
    assertEquals("Apfel", paar.gibDeutschesWort());

    paar = paar.gibNächstes();
    assertEquals("Birne", paar.gibDeutschesWort());

    paar = paar.gibNächstes();
    assertEquals("Klementine", paar.gibDeutschesWort());

    paar = paar.gibNächstes();
    assertEquals("Wassermelone", paar.gibDeutschesWort());

    paar = paar.gibNächstes();
    assertEquals("Zitrone", paar.gibDeutschesWort());
  }
}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/aufgaben/aud/listen/woerterbuch/WoerterbuchTest.java`

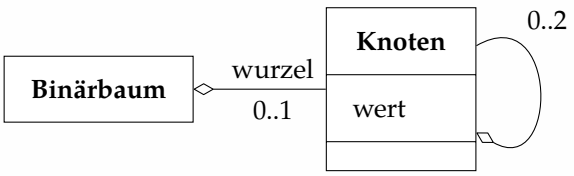
Binärbaum
Klassendiagramm

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

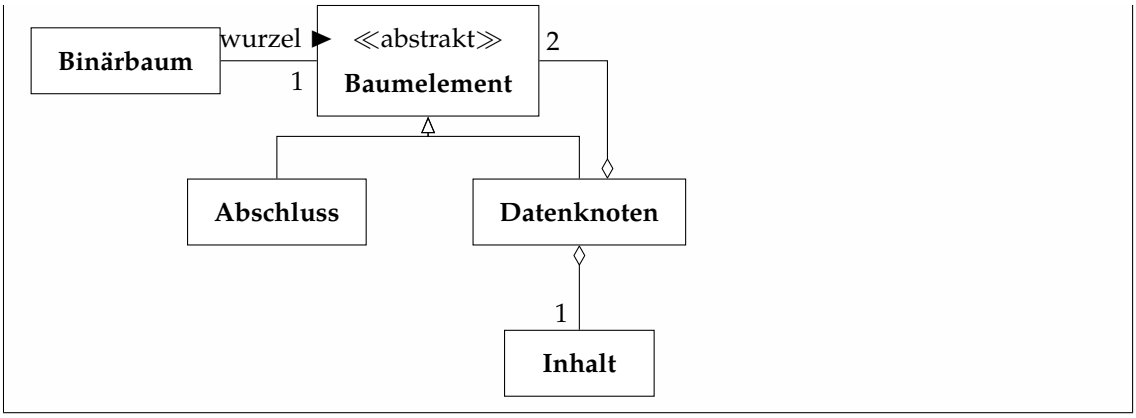
(a) Erstellen Sie ein Klassendiagramm für einen Binärbaum.

Lösungsvorschlag

einfacher Binärbaum



Binärbaum mit Kompositum



(b) Entwerfen Sie eine mögliche Implementierung zur Erzeugung eines binären Baumes in Java.

Lösungsvorschlag

einfacher Binärbaum

```
public class Binaerbaum {
  public Knoten wurzel;

  public void fügeEin(int wert) {
    if (wurzel == null) {
      wurzel = new Knoten(wert);
    } else {
      if (wert <= wurzel.gibWert()) {
        if (wurzel.gibLinks() != null) {
          wurzel.gibLinks().fügeEin(wert);
        } else {
          wurzel.setzeLinks(new Knoten(wert));
        }
      } else {
        if (wurzel.gibRechts() != null) {
          wurzel.gibRechts().fügeEin(wert);
        } else {
          wurzel.setzeRechts(new Knoten(wert));
        }
      }
    }
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/einfach/Binaerbaum.java`

```
public class Knoten {

  private Knoten links;
  private Knoten rechts;
  private int wert;

  public Knoten(int wert) {
```

```
    this.wert = wert;
  }

  public void setzeWert(int w) {
    wert = w;
  }

  public int gibWert() {
    return wert;
  }

  public void setzeLinks(Knoten l) {
    links = l;
  }

  public void setzeRechts(Knoten r) {
    rechts = r;
  }

  public Knoten gibLinks() {
    return links;
  }

  public Knoten gibRechts() {
    return rechts;
  }

  public void fügeEin(int wert) {
    if (wert <= this.gibWert()) {
      if (this.gibLinks() != null) {
        this.gibLinks().fügeEin(wert);
      } else {
        this.setzeLinks(new Knoten(wert));
      }
    } else {
      if (this.gibRechts() != null) {
        this.gibRechts().fügeEin(wert);
      } else {
        this.setzeRechts(new Knoten(wert));
      }
    }
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/einfach/Knoten.java`

Binärbaum mit Kompositum

```
class Binaerbaum {
  private Baumelement wurzel;
```

```
  public Binaerbaum() {
    wurzel = new Abschluss();
  }

  public void setzeWurzel(Baumelement wurzel) {
    this.wurzel = wurzel;
  }

  public Baumelement gibWurzel() {
    return wurzel;
  }

  public int gibAnzahl() {
    return wurzel.gibAnzahl();
  }

  public void gibAus() {
    wurzel.gibAus();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Binaerbaum.java`

```
abstract class Baumelement {
  public abstract void setzteLinks(Baumelement nl);

  public abstract void setzeRechts(Baumelement nr);

  public abstract Baumelement gibLinks();

  public abstract Baumelement gibRechts();

  public abstract Datenelement gibInhalt();

  public abstract int gibAnzahl();

  public abstract void gibAus();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Baumelement.java`

```
class Abschluss extends Baumelement {

  public void setzteLinks(Baumelement links) {
    System.out.println("Ein Abschluss hat kein linkes Element!");
  }

  public void setzeRechts(Baumelement rechts) {
    System.out.println("Ein Abschluss hat kein rechts Element!");
  }

  public Baumelement gibLinks() {
    System.out.println("Linkes Element nicht bekannt!");
    return this;
```

```
  }

  public Baumelement gibRechts() {
    System.out.println("Linkes Element nicht bekannt!");
    return this;
  }

  public Datenelement gibInhalt() {
    return null;
  }

  public int gibAnzahl() {
    return 0;
  }

  public void gibAus() {
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Abschluss.java`

```
class Datenknoten extends Baumelement {
  private Baumelement links, rechts;
  private Datenelement inhalt;

  public Datenknoten(Baumelement links, Baumelement rechts, Datenelement
↪   inhalt) {
    this.links = links;
    this.rechts = rechts;
    this.inhalt = inhalt;
  }

  public void setzteLinks(Baumelement links) {
    this.links = links;
  }

  public void setzeRechts(Baumelement rechts) {
    this.rechts = rechts;
  }

  public void inhaltSetzen(Datenelement inhalt) {
    this.inhalt = inhalt;
  }

  public Baumelement gibLinks() {
    return links;
  }

  public Baumelement gibRechts() {
    return rechts;
  }

  public Datenelement gibInhalt() {
    return inhalt;
```

```
  }

  public int gibAnzahl() {
    return 1 + links.gibAnzahl() + rechts.gibAnzahl();
  }

  public void gibAus() {
    System.out.print(" [");
    links.gibAus();
    inhalt.gibAus();
    rechts.gibAus();
    System.out.print("] ");
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Datenknoten.java`

```
abstract class Datenelement {
  public abstract void gibAus();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Datenelement.java`

```
class Inhalt extends Datenelement {
  private String inhalt;

  public Inhalt(String inhalt) {
    this.inhalt = inhalt;
  }

  public String gibInhalt() {
    return inhalt;
  }

  public void gibAus() {
    System.out.print(inhalt);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/baum/kompositum/Inhalt.java`

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class BinaerbaumTest {

  @Test
  public void teste(){
    Binaerbaum baum = new Binaerbaum();
    Inhalt[] inhalte = new Inhalt[16];
    Datenknoten[] datenknoten = new Datenknoten[16];
    inhalte[0] = new Inhalt("Inhalt 1");
```

```
    inhalte[1] = new Inhalt("Inhalt 2");
    inhalte[2] = new Inhalt("Inhalt 3");

    inhalte[3] = new Inhalt("Inhalt 4");
    inhalte[4] = new Inhalt("Inhalt 5");

    for (int i = 0; i < 5; i++) {
      datenknoten[i] = new Datenknoten(new Abschluss(), new Abschluss(),
↪   inhalte[i]);
    }
    baum.setzeWurzel(datenknoten[0]);

    datenknoten[0].setzteLinks(datenknoten[1]);
    datenknoten[0].setzeRechts(datenknoten[2]);

    datenknoten[1].setzteLinks(datenknoten[3]);
    datenknoten[1].setzeRechts(datenknoten[4]);

    assertEquals(5, baum.gibAnzahl());

    Inhalt inhalt = (Inhalt)
↪   baum.gibWurzel().gibLinks().gibLinks().gibInhalt();
    assertEquals("Inhalt 4", inhalt.gibInhalt());
  }

}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/aufgaben/aud/baum/kompositum/BinaerbaumTest.java`

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

(a) Beschreiben Sie den Unterschied zwischen einem Zustands- und einem Aktivitätsdiagramm.

Lösungsvorschlag

Das Aktivitätsdiagramm zeigt den Fluss von Aktivitäten innerhalb des Systems. Das Zustandsdiagramm hingegen beschreibt die Reaktion eines Systems auf Ereignisse.

(b) Der DVD-Automat für die Filmauswahl aus Blatt 2, Aufgabe 1 soll als Zustandsdiagramm modelliert werden. Beachten Sie dabei die angegebenen Funktionalitäten des Automaten.

(i) Geben Sie die Ein- und Ausgaben des Automaten für die Filmauswahl und für die Aus- und Rückgabe von Filmen an.

Lösungsvorschlag

Eingaben: Kundenkarte, Fingerabdruck, DVDs, Suchbegriff

Ausgaben: Kundenkarte, DVDs, Bildschirmausgaben (Filmtitel, Regisseur, Hauptdarsteller, Kurzbeschreibung, Erscheinungsjahr, Alters-

freigabe, Verfügbarkeit, Fehlermeldung etc.)

(ii) Geben Sie alle Zustandsattribute an, die für die Modellierung der Automaten notwendig sind und beschreiben Sie deren Verwendungszweck.

(iii) Identifizieren Sie anhand der Zustandsattribute die Zustände der obigen Automaten und geben Sie eine Charakterisierung der Zustände durch Angabe der möglichen Wertebereiche der Zustandsattribute an. Welcher der Zustände ist der Anfangszustand?

(iv) Zeichnen Sie die Zustandsübergangsdiagramme. Verwenden Sie hierzu die Syntax mit Ein- und Ausgabe, Vor- und Nachbedingungen.

(c) Betrachten wir nun einen gewöhnlichen Video- und DVD-Verleih. Beschreiben Sie das Szenario des Ausleihs eines Videos mit Hilfe eines Aktivitätsdiagramms. Bei dem Videoverleih gelte:

- Der Kunde identifiziert sich beim Ausleihen mit seiner Kundenkarte oder seinem Passwort. Hat der Kunde noch keine Karte, so muss der Mitarbeiter ihn registrieren und ihm eine Kundenkarte ausstellen.
- Filme können wie beim Automaten per Internet bis zu zwei Stunden im Voraus reserviert werden.
- Der Kunde hat kein Gehaltskonto, sondern bezahlt seine Gebühren bei der Rückgabe des Videos in bar oder per Karte.
- Ansonsten gelten die gleichen Bedingungen wie beim Automaten.

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

Stellen Sie sich vor, Sie brauchen ein Grafiksystem. In diesem System wollen Sie aus Linien, Rechtecken, und Text größere Abbildungen darstellen. Diese Abbildungen sollen auch wieder andere Abbildungen rekursiv enthalten können. Sie brauchen also primitive Objekte: die Linie, das Rechteck und den Text. Zusätzlich brauchen Sie Behälter, die weitere Behälter und primitive Objekte enthalten können.

Erklären Sie, mit welchem Entwurfsmuster man diese Struktur abbilden kann und zeichnen Sie das dazugehörige Klassendiagramm.

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 2

(a) Erklären Sie, was man bei der Entwicklung von Softwaresystemen unter einem Entwurfsmuster versteht und gehen Sie dabei auch auf die Vorteile ein.

Lösungsvorschlag

Entwurfsmuster sind bewährte Lösungsschablonen für wiederkehrende Entwurfsprobleme sowohl in der Architektur als auch in der Softwarearchitektur und -entwicklung. Sie stellen damit eine wiederverwendbare Vorlage zur Problemlösung dar, die in einem bestimmten Zusammenhang einsetzbar ist.

Der primäre Nutzen eines Entwurfsmusters liegt in der Beschreibung einer Lösung für eine bestimmte Klasse von Entwurfsproblemen. Weiterer Nutzen

ergibt sich aus der Tatsache, dass jedes Muster einen Namen hat. Dies vereinfacht die Diskussion unter Entwicklern, da man abstrakt über eine Struktur sprechen kann. So sind etwa Software-Entwurfsmuster zunächst einmal unabhängig von der konkreten Programmiersprache. Wenn der Einsatz von Entwurfsmustern dokumentiert wird, ergibt sich ein weiterer Nutzen dadurch, dass durch die Beschreibung des Musters ein Bezug zur dort vorhandenen Diskussion des Problemkontextes und der Vor- und Nachteile der Lösung hergestellt wird.

Warteschlange (Queue)
Klassendiagramm

(b) Nennen Sie die drei ursprünglichen Typen von Entwurfsmuster, erklären Sie diese kurz und geben Sie zu jedem Typ drei Beispiele an.

Lösungsvorschlag

Typ	**Erlärung**	**Beispiele**
Erzeugungsmuster	Dienen der Erzeugung von Objekten; diese wird dadurch gekapselt und ausgelagert, um den Kontext der Objekterzeugung unabhängig von der konkreten Implementierung zu halten	abstrakte Fabrik, Singleton, Prototyp
Strukturmuster	Erleichtern den Entwurf von Software durch vorgefertigte Schablonen für Beziehungen zwischen Klassen.	Adapter, Kompositum, Stellvertreter
Verhaltensmuster	Modellieren komplexes Verhalten der Software und erhöhen damit die Flexibilität der Software hinsichtlich ihres Verhaltens.	Beobachter, Interpreter, Iterator

46116 / 2010 / Frühjahr / Thema 1 / Aufgabe 1

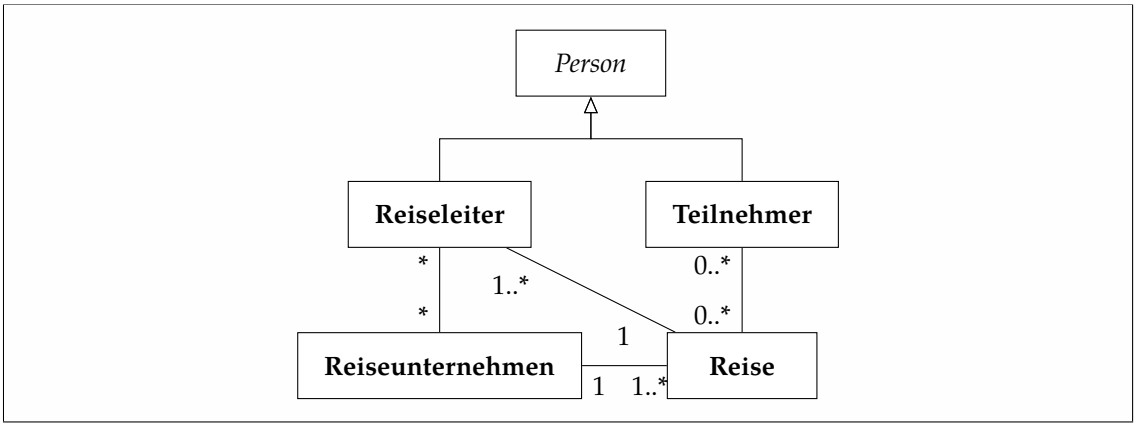
Es sei folgender Sachverhalt gegeben:

Ein Reiseunternehmen bietet verschiedene Reisen an. Dazu beschäftigt es eine Reihe von Reiseleitern, wobei eine Reise von mindestens einem Reiseleiter geleitet wird. Da Reiseleiter freiberuflich arbeiten, können sie bei mehreren Reiseunternehmen Reisen leiten.

An einer Reise können mehrere Teilnehmer teilnehmen, ein Teilnehmer kann auch an verschiedenen Reisen teilnehmen.

(a) Modellieren Sie diesen Sachverhalt in einem UML-Klassendiagramm. Für Teilnehmer und Reiseleiter sollen Sie dabei eine abstrakte Oberklasse definieren. Achten Sie dabei auf die Multiplizitäten der Assoziationen. Sie müssen keine Attribute bzw. Methoden angeben.

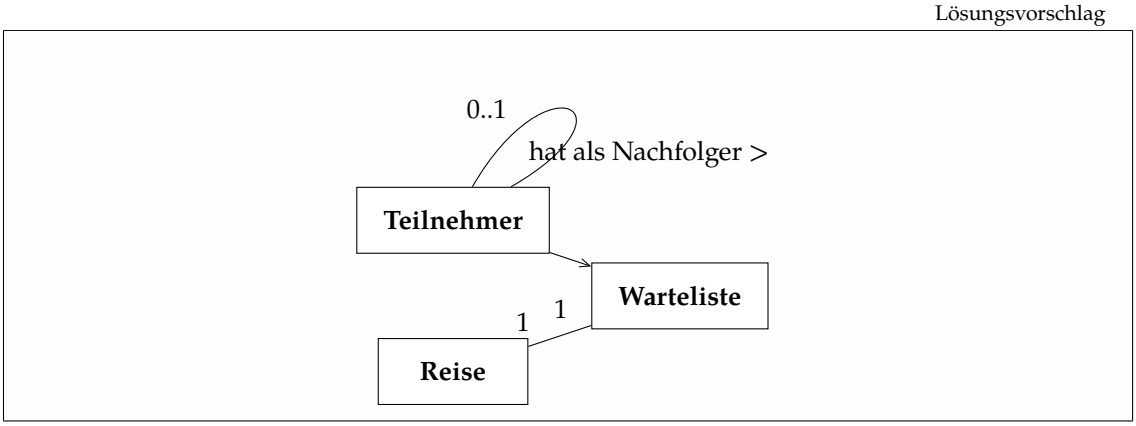
Lösungsvorschlag Implementierung in Java



(b) Eine Reise kann jedoch nur mit einer begrenzten Kapazität angeboten werden, das heißt, zu einer bestimmten Reise kann nur eine begrenzte Anzahl von Teilnehmern assoziiert werden. Als Ausgleich soll pro Reise eine Warteliste verwaltet werden.

Modellieren Sie diesen erweiterten Sachverhalt in einem neuen Diagramm. Nicht veränderte Klassen brauchen nicht noch einmal angegeben werden. Beachten Sie dabei, dass die Reihenfolge bei einer Warteliste eine Rolle spielt.

Lösungsvorschlag



(c) Implementieren Sie die in Aufgabenteil b) modellierten Klassen in Java. Fügen Sie eine Methode hinzu, die einen Teilnehmer von einer Reise entfernt. Dabei soll automatisch der erste Platz der Warteliste zu einem Reiseteilnehmer werden, wenn die Warteliste nicht leer ist. Achten Sie auf die Navigierbarkeit Ihrer Assoziationen. Sie können davon ausgehen, dass die Methode nur mit Teilnehmern aufgerufen wird, die in der Tat Teilnehmer der Reise sind.

```
public class Teilnehmer {
  Teilnehmer nächster;

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/Teilnehmer.java`

```
/**
 * Diese Klasse ist eine Implementation einer einfach verketteten Liste. Sie
```

```
 * wird einerseit in der Klasse {@link Reise} genutzt, um die Reiseteilnehmer zu
 * speichern, andererseits um eine Warteliste darauf aufbauen zu könnten.
 */
public class TeilnehmerListe {

}
```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/TeilnehmerListe.java

```
/**
 * Eine Reise kann jedoch nur mit einer begrenzten Kapazität angeboten
 * werden, das heißt, zu einer bestimmten Reise kann nur eine begrenzte
 * Anzahl von Teilnehmern assoziiert werden. Als Ausgleich soll pro
 * Reise eine Warteliste verwaltet werden.
 *
 * Modellieren Sie diesen erweiterten Sachverhalt in einem neuen
 * Diagramm. Nicht veränderte Klassen brauchen nicht noch einmal
 * angegeben werden. Beachten Sie dabei, dass die Reihenfolge bei einer
 * Warteliste eine Rolle spielt.
 *
 * Implementieren Sie die in Aufgabenteil b) modellierten Klassen in
 * Java. Fügen Sie eine Methode hinzu, die einen Teilnehmer von einer
 * Reise entfernt. Dabei soll automatisch der erste Platz der Warteliste
 * zu einem Reiseteilnehmer werden, wenn die Warteliste nicht leer ist.
 * Achten Sie auf die Navigierbarkeit Ihrer Assoziationen. Sie können
 * davon ausgehen, dass die Methode nur mit Teilnehmern aufgerufen wird,
 * die in der Tat Teilnehmer der Reise sind.
 */
public class Warteliste extends TeilnehmerListe {

}
```

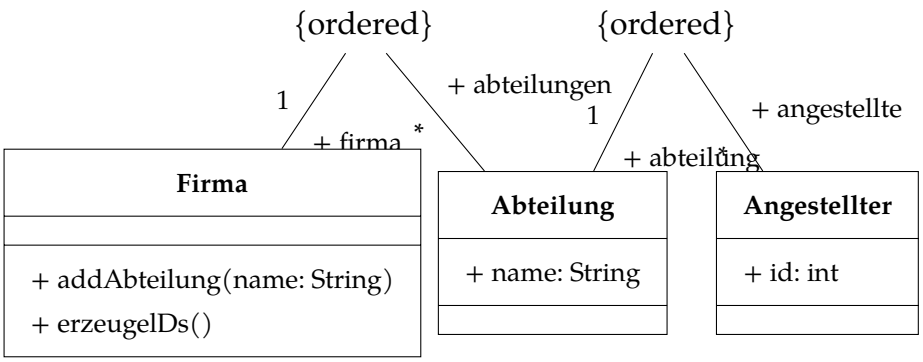
Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/Warteliste.java

```
public class Reise {


  Teilnehmer teilnehmer;

  Warteliste warteliste;

  void entferneTeilnehmer(Teilnehmer teilnehmer) {

  }

}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46116/jahr_2010/fruehjahr/reiseunternehmen/Reise.java

46116 / 2011 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1

Firmenstruktur

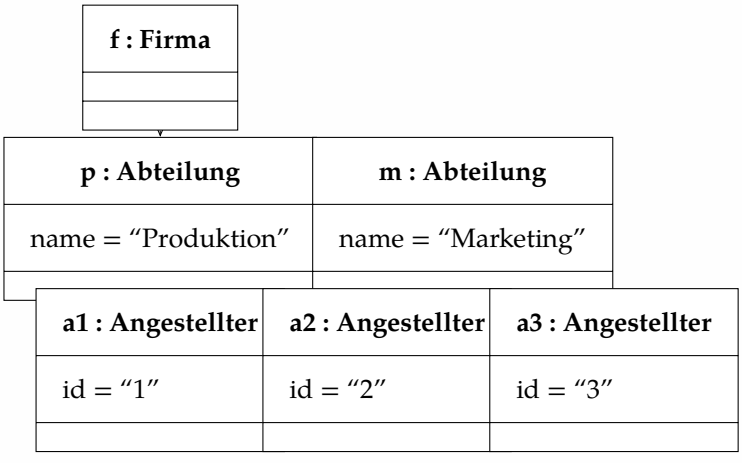


Eine Firma besteht aus `null` oder mehr Abteilungen, von denen jede `null` oder mehr Angestellte hat. Da sowohl die Abteilungen als auch deren Angestellten geordnet sind, sind Angestellte insgesamt geordnet. Sie haben durchgehende, ganzzahlige IDs, die bei 1 beginnen.

(a) Erstellen Sie exemplarisch ein Objektdiagramm: Stellen Sie eine Firma mit dem Instanznamen `f` und den zwei Abteilungen „Produktion“ (Name `p`) und „Marketing“ (Name `m`) dar. Die Produktion hat zwei Angestellte, Marketing hat einen Angestellten. Die Angestellten haben die Namen `a1`, `a2`, und `a3`.

Lösungsvorschlag

Die Instanzbezeichnung müsste noch unterstrichen werden. Das geht aber leider mit TikZ-UML nicht.



(b) Implementieren Sie das Klassendiagramm in Java oder in einer anderen geeigneten objektorientierten Programmiersprache Ihrer Wahl. Beachten Sie, dass die Assoziationen bidirektional und geordnet sind. Die beiden Methoden der Klasse `Firma` sollen dabei folgendes Verhalten haben:

Die Methode `erzeugeIDs` sorgt dafür, dass die IDs wieder korrekt zugewiesen sind. Die alten IDs können beliebig geändert werden, solange das Endergebnis wieder den obenstehenden Kriterien genügt.

Lösungsvorschlag

```
import java.util.ArrayList;
import java.util.List;

public class Firma {

  List<Abteilung> abteilungen;

  public Firma() {
    abteilungen = new ArrayList<Abteilung>();
  }

  public void addAbteilung(String name) {
    for (Abteilung abteilung : abteilungen) {
      if (abteilung.name.equals(name)) {

↪   System.out.println("Eine Abteilung mit diesem Namen gibt es bereits schon.");
        return;
      }
      abteilungen.add(new Abteilung(name));
    }
  }

  public void erzeugeIDs() {
    int idZähler = 1;
    for (Abteilung abteilung : abteilungen) {
      for (Angestellter angestellter : abteilung.angestellte) {
        angestellter.id = idZähler;
        idZähler++;
      }
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2011/fruehjahr/Firma.java`

```
import java.util.ArrayList;
import java.util.List;

public class Abteilung {
  public String name;

  public List<Angestellter> angestellte;

  public Abteilung(String name) {
    this.name = name;
    angestellte = new ArrayList<Angestellter>();
  }
}
```

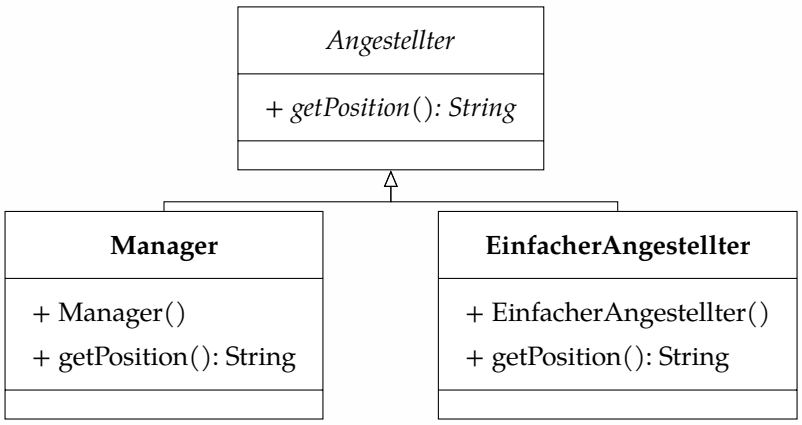
Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2011/fruehjahr/Abteilung.java`

```java
import java.util.List;

public class Angestellter {
  public int id;

  public Firma firma;

  public List<Angestellter> angestellte;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2011/fruehjahr/Angestellter.java`

(c) Angestellte sollen in Manager und einfache Angestellte unterteilt werden. Zeichnen Sie ein Klassendiagramm mit der Oberklasse `Angestellter` und den zwei Unterklassen `Manager` und `EinfacherAngestellter`. Die Klasse `Angestellter` soll nicht instantiierbar sein und erzwingen, dass die Methode `getPosition()` (öffentlich, ohne Argumente, Rückgabewert `String`) von allen konkreten Unterklassen implementiert wird. `Manager` und `EinfacherAngestellter` sollen instantiierbar sein.

Lösungsvorschlag



(d) Wie lautet der Fachbegriff dafür, dass eine Methode in einer Klasse und in deren Unterklassen dieselbe Signatur hat, aber in den Unterklassen unterschiedlich implementiert ist?

Lösungsvorschlag

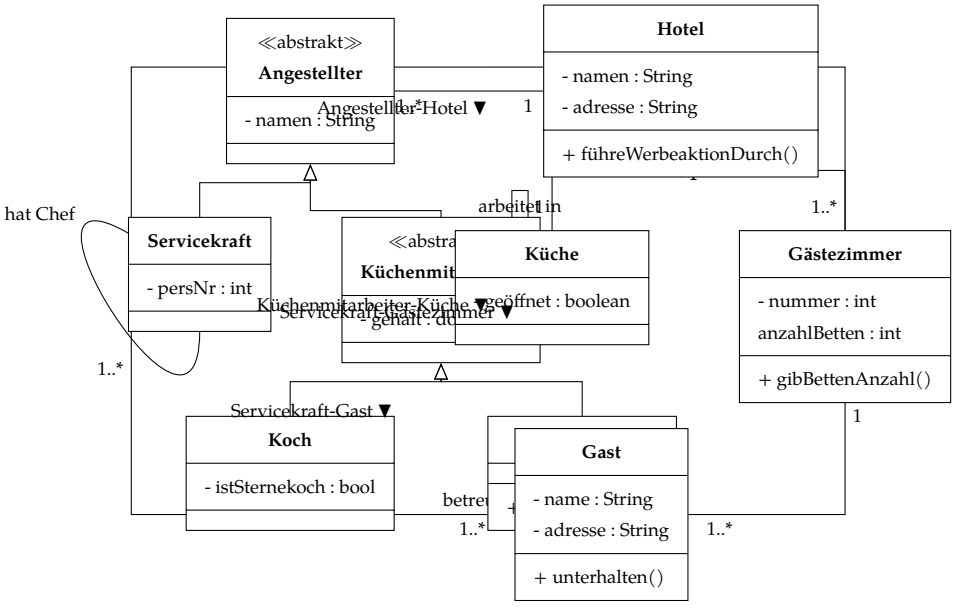
Abstrakte Methode

46116 / 2012 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 1

Hotel-Verwaltung

Erstellen Sie zu der folgenden Beschreibung eines Systems zur Organisation eines Hotels ein Klassendiagramm, das Attribute und Assoziationen mit Kardinalitäten sowie Methoden enthält. Setzen Sie dabei das Konzept der Vererbung sinnvoll ein.

- Ein **Hotel** besteht aus genau einer Küche und mehreren Gästezimmern.
- Es hat einen Namen, eine Adresse und kann Werbeaktionen durchführen.
- Alle Mitarbeiter, sowohl Servicekräfte als auch Küchenmitarbeiter, sind Angestellte des Hotels.
- Die Küche kann geöffnet oder geschlossen sein.
- Gästezimmer haben eine Nummer und eine bestimmte Anzahl an Betten, die ausgegeben werden kann.
- In diesen Gästezimmern wohnen Gäste, die von einer oder mehreren Servicekräften umsorgt werden.
- Servicekräfte sind nur für die ihnen zugeordneten Gästezimmer verantwortlich.
- Jede Servicekraft hat einen Namen und eine Personalnummer sowie einen Vorgesetzten, der auch eine Servicekraft ist.
- In der Küche arbeiten Küchenmitarbeiter, die einen Namen haben und ein Gehalt bekommen.
- Küchenmitarbeiter sind entweder Köche oder Aushilfen. Köche können zudem Sterneköche sein, also mit einem oder mehreren Sternen dekoriert sein.
- Aushilfen bauen die Büffets für die Mahlzeiten auf.
- Gäste können sich unterhalten. Jeder Gast hat einen Namen und eine Adresse und ist seinem Zimmer zugeordnet.



46116 / 2013 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 1

Klassendiagramm

Aufgabe 1

Gegeben sei folgendes Klassendiagramm:

(a) Implementieren Sie die Klassen „Order“, „Payment“, „Check“, „OrderDetail“ und „Item“ in einer geeigneten objektorientierten Sprache Ihrer Wahl. Beachten Sie dabei insbesondere Sichtbarkeiten, Klassen- vs. Instanzzugehörigkeiten und Schnittstellen bzw. abstrakte Klassen.

Lösungsvorschlag

```java
public class Check extends Payment {
  String bankName;
  long bankId;
  public boolean authorized() {
    return true;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2013/fruehjahr/t2_ta1_a1/Check.java`

```java
@SuppressWarnings("unused")
public class Item {
  private double weight;
  public String description;

  public double getPrice() {
    return 42;
  }

  public double getWeight() {
    return 23;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2013/fruehjahr/t2_ta1_a1/Item.java`

```java
import java.util.Date;

public class Order {
  public static double VAT = 0.19;

  protected Date date;

  protected boolean shipped;

  public double calcPrice() {
    return 0.1d;
  }

  public double calcWeight() {
    return 0.2d;
```

```
  }

  public double calcTax(double tax) {
    return 0.3d;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2013/fruehjahr/t2_ta1_a1/Order.java`

```
@SuppressWarnings("unused")
public class OrderDetail {
  private long quantity;

  Item item;

  public double calcPrice() {
    return 0.1d;
  }

  public double calcWeight() {
    return 0.2d;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2013/fruehjahr/t2_ta1_a1/OrderDetail.java`

```
public abstract class Payment {
  double amount;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2013/fruehjahr/t2_ta1_a1/Payment.java`

(b) Erstellen Sie ein Sequenzdiagramm (mit konkreten Methodenaufrufen und Numerierung der Nachrichten) für folgendes Szenario:

(i) „Erika Mustermann“ aus „Rathausstraße 1, 10178 Berlin“ wird als neue Kundin angelegt.

(ii) Frau Mustermann bestellt heute 1 kg Gurken und 2 kg Kartoffeln.

(iii) Sie bezahlt mit ihrer Visa-Karte, die im August 2014 abläuft und die Nummer „1234 567891 23456“ hat — die Karte erweist sich bei der Prüfung als gültig.

(iv) Am Schluss möchte sie noch wissen, wie viel ihre Bestellung kostet — dabei wird der Anteil der Mehrwertsteuer extra ausgewiesen.

46116 / 2014 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 3

Gegeben sei folgender Sachverhalt:

Für eine Verwaltungssoftware einer Behörde soll ein Bestellsystem entwickelt werden. Dabei sollen die Nutzer ihre Raummaße eingeben können. Anschließend können die Nutzer über ein Web-Interface das Büro gestalten und Möbel (wie zum Beispiel

Anwendungsfalldiagramm

Wandschränke) und andere Einrichtungsgegenstände in einem virtuellen Büro platzieren. Aus dem Web-Interface kann die Einrichtung dann direkt bestellt werden. Dazu müssen die Nutzer ihre Büro-Nummer und den Namen und die Adresse der Behörde eingeben und die Bestellung bestätigen.

Weiterhin können Nutzer auch Büromaterialien über das Web-Interface bestellen. Dazu ist anstatt der Eingabe der Raummaße nur das Eingeben von Büro-Nummer und des Namens und der Adresse der Behörde erforderlich.

Zusätzlich zum Standard-Nutzer können sich auch Administratoren im System anmelden und Möbel zur Kollektion hinzufügen und aus der Kollektion entfernen. Die Möbel können eindeutig durch ihre Inventurnummer identifiziert werden.

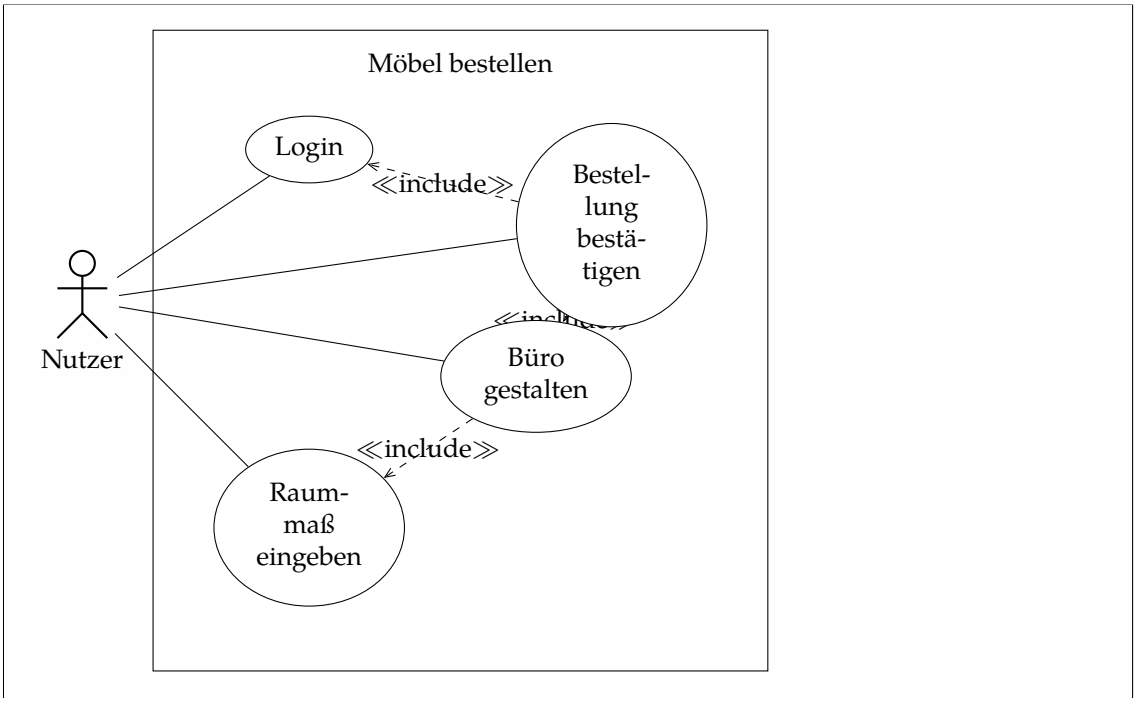
Um jegliche Veränderungen im System protokollieren zu können müssen Nutzer und Administratoren zur Bestätigung eingeloggt sein.

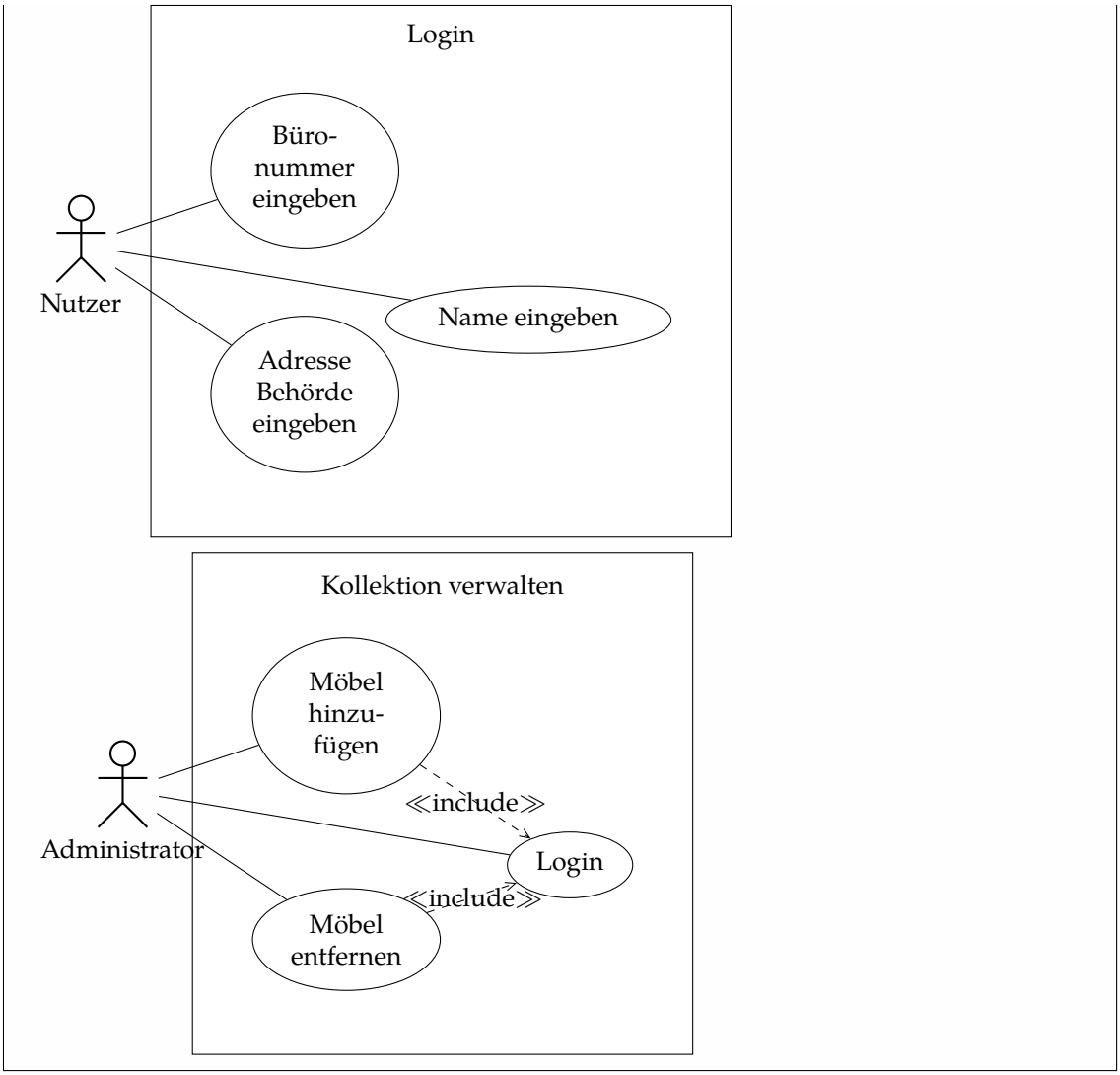
(a) Erfassen Sie die drei Systemfunktionen *Möbel bestellen*, *Login* und *Kollektion verwalten* in einem UML-konformen Use Case Diagramm.

Login

Kollektion verwalten

Lösungsvorschlag





(b) Erstellen Sie ein UML-Klassendiagramm, welches die Beziehungen und sinnvolle Attribute der Klassen „Nutzer, Büro, Möbelstück und Wandschrank“ darstellt.

(c) Instanziieren Sie das Klassendiagramm in einem Objektdiagramm mit den zwei Nutzern mit Namen Ernie und Bernd, einem Büro mit der Nummer CAPITOL2 und zwei Schränken mit den Inventurnummern S1.88 und S1.77.

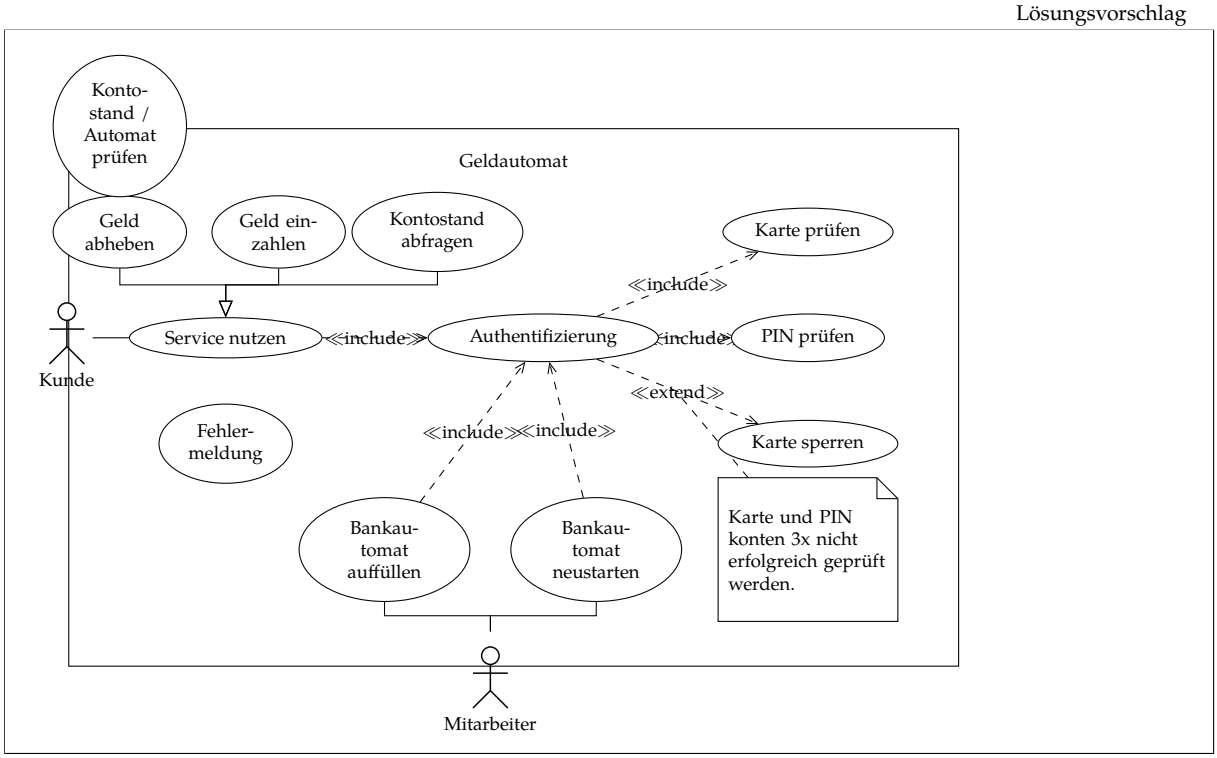
(d) Geben Sie für ein Büromöbelstück ein Zustandsdiagramm an. Überlegen Sie dazu, welche möglichen Zustände ein Möbelstück während des Bestellvorgangs haben kann und finden Sie geeignete Zustandsübergänge.

46116 / 2017 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 2

Im Folgenden sehen Sie ein fehlerhaftes Use-Case-Diagramm für das System „Geldautomat”. Geben Sie ein korrektes Use-Case-Diagramm (Beziehungen und Beschriftungen) entsprechend der folgenden Beschreibung an. Sollte es mehrere Möglichkeiten geben, begründen Sie Ihre Entscheidung kurz.

Kunden können am Geldautomat verschiedene Services nutzen, es kann Geld abgehoben und eingezahlt sowie der Kontostand abgefragt werden. Für jeden dieser Services ist eine Authentifizierung notwendig. Diese besteht aus der Prüfung der Bankkarte und des eingegebenen PINs. Manche Bankautomaten können Karten bei zu vielen Fehlversuchen (3) bei der Anmeldung sperren, andere geben Fehlermeldungen aus, falls die Karte gesperrt wurde oder nicht mehr genügend Geld auf dem Konto oder im Bankautomaten vorhanden ist. Mitarbeiter bzw. Mitarbeiterinnen der Bank können sich ebenfalls über PIN und Karte authentifizieren, um dann den Bankautomaten neu zu starten oder mit Geld aufzufüllen. Bevor Geld abgehoben werden kann, ist sicherzustellen, dass auf dem Konto und im Bankautomaten genügend Geld vorhanden ist.

Lösungsvorschlag



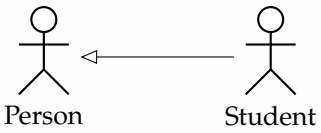
46116 / 2017 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 3

Betrachten Sie die folgenden UML-Diagramme. Sind diese korrekt? Falls nein, begründen Sie warum nicht. Geben Sie in diesem Fall außerdem eine korrigierte Version an. Falls ja, erklären Sie die inhaltliche Bedeutung des Diagramms.

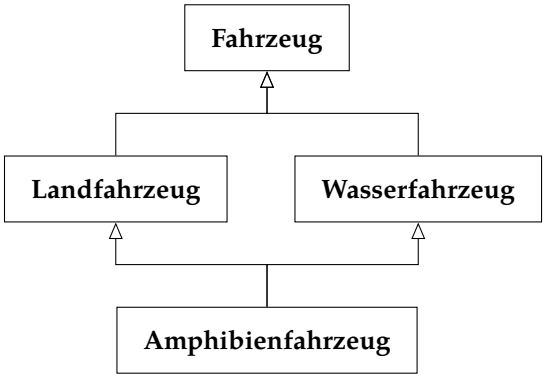
(a)

Lösungsvorschlag

Falsch, den verwendeten „extends“-Pfeil gibt es nur zwischen Anwendungsfällen.



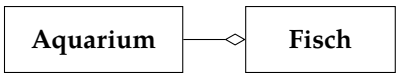
(b)



Lösungsvorschlag

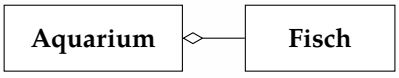
Die dargestellte Modellierung ist *korrekt*. Sowohl Land- als auch Wasserfahrzeuge sind Fahrzeuge und erben somit von dieser Klasse. Da ein Amphibienfahrzeug eine „Mischung“ aus beidem ist, erbt diese Klasse auch von beiden Klassen. Diese Mehrfachvererbug kann allerdings nicht in jeder Programmiersprache (z. B. nicht in Java) umgesetzt werden.

(c)

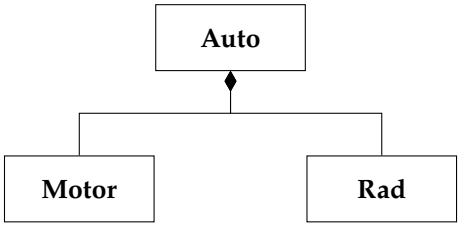


Lösungsvorschlag

Bei der dargestellten Aggregation befindet sich die Raute an der *falschen* Seite der Beziehung. Das Diagramm würde bedeuten, dass ein Fisch mehrere Aquarien enthält. Die Umkehrung ist aber korrekt:

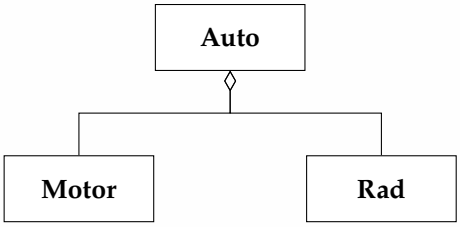


(d)



Lösungsvorschlag

Hier wurde für die Modellierung eine Komposition gewählt. Dies bedeutet, dass die Existenz der Teile vom Ganzen abhängt. Einen Raum kann es z. B. ohne ein Gebäude nicht geben. In diesem Fall ist die Darstellung *falsch*, da Motor und Rad auch ohne Auto existieren können. Die Modellierung muss also mittels Aggregation erfolgen:



46116 / 2017 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 3

Verwenden Sie geeignete **Entwurfsmuster**, um die folgenden Sachverhalte mit Hilfe von **UML-Klassendiagrammen** zu beschreiben. Nennen Sie das zu verwendende Entwurfsmuster namentlich, wenden Sie es zur Lösung derjeweiligen *Fragestellung* an und erstellen Sie damit das problemspezifische UML-Klassendiagramm. Beschränken Sie sich dabei auf die statische Sicht, ðdefinieren Sie keinerlei Verhalten mit Ausnahme der Definition geeigneter Operationen.

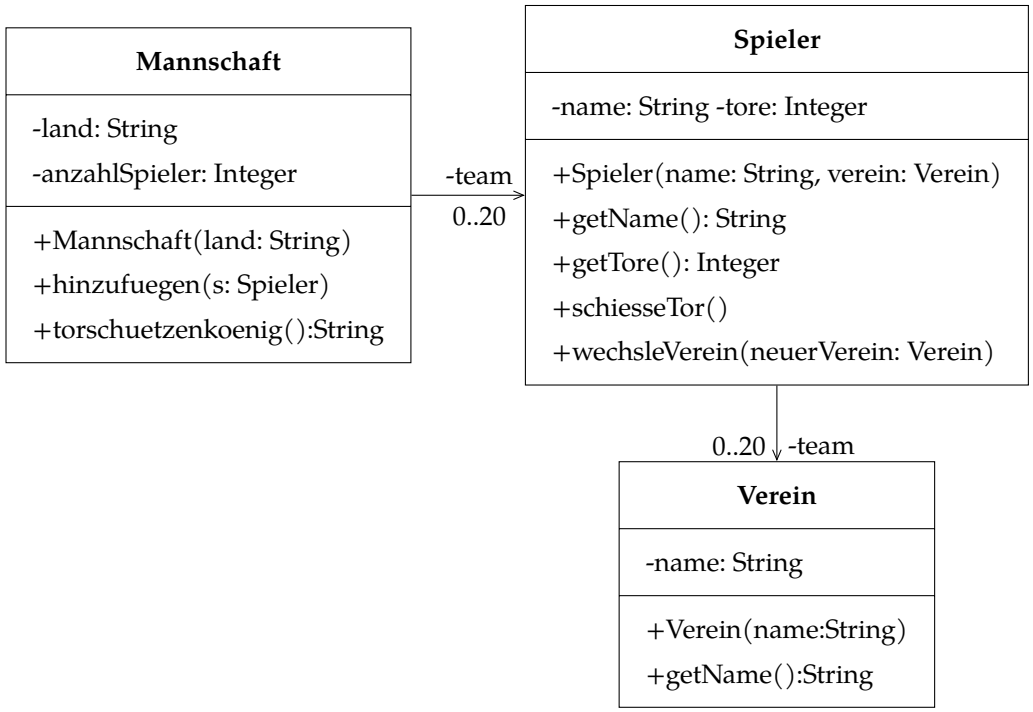
(a) Es gibt unterschiedliche Arten von Bankkonten: Girokonto, Bausparkonto vmd Kreditkarte. Bei allen Konten ist der Name des Inhabers hinterlegt. Girokonten haben eine IBAN. Kreditkarten sind immer mit einem Girokonto verknüpft. Bei Bausparkonten werden ein Sparzins sowie ein Darlehenszins festgelegt. Es gibt eine *zentrale* Klasse, die die *Erzeugung* unterschiedlicher Typen von Bankkonten steuert.

(b) Beim Ticker für ein Hockeyspiel können sich verschiedene Geräte registrieren und wieder abmelden, um auf *Veränderungen* des Spielstands *zu reagieren*. Hierzu werden im Ticker die Tore der Heim- vmd Gastmannschaft sowie die aktuelle Spielminute vermerkt. Als konkrete Geräte sind eine Smartphone-App sowie eine Stadionuhr bekannt.

(c) Dateisysteme sind *baumartig* strukturiert. Verzeichnisse können wiederum selbst Verzeichnisse und/oder Dateien beinhalten. Sowohl Dateien als auch Verzeichnisse haben einen Namen.Das jeweilige Eltemverzeichnis ist eindeutig. Bei Dateien wird die Art(Binär, Text oder andere) sowie die Größe in Byte, bei Verzeichnissen die Anzahl enthaltener Dateien hinterlegt.

46116 / 2018 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 3

Für die nächste Fußballweltmeisterschaft möchte ein Wettbüro ein Programm zur Verwaltung von Spielern, Vereinen und (National-)Mannschaften entwickeln. Dazu wurde bereits das folgende UML-Klassendiagramm entworfen.



Es kann angenommen werden, dass die Klasse `Verein` bereits implementiert ist. In den folgenden Implementierungsaufgaben können Sie eine objektorientierte Programmiersprache Ihrer Wahl verwenden. Die verwendete Sprache ist anzugeben. Zu beachten sind jeweils die im Klassendiagramm angegebenen Sichtbarkeiten von Attributen, Rollennamen, Konstruktorn und Operationen.

(a) Es ist eine Implementierung der Klasse `Spieler` anzugeben. Der Konstruktor soll die Instanzvariablen mit den gegebenen Parametern initialisieren, wobei die. Anzahl der Tore nach der Objekterzeugung gleich 0 sein soll. Ansonsten kann die Funktionalität der einzelnen Operationen aus deren Namen geschlossen werden.

Lösungsvorschlag

```java
public class Spieler {
  private String name;
  private int tore;
  @SuppressWarnings("unused")
  private Verein verein;

  public Spieler(String name, Verein verein) {
    this.name = name;
    this.verein = verein;
    tore = 0;
  }

  public String getName() {
    return name;
  }

  public int getTore() {
    return tore;
  }
```

```
  public void schiesseTor() {
    tore++;
  }

  public void wechsleVerein(Verein neuerVerein) {
    verein = neuerVerein;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2018/fruehjahr/verein/Spieler.java`

(b) Es ist eine Implementierung der Klasse `Mannschaft` anzugeben. Der Konstruktor soll das Land initialisieren, die Anzahl der Spieler auf 0 setzen und das Team mit einem noch „leeren “Array der Länge 20 initialisieren. Das Team soll mit der Methode `hinzufuegen` um einen Spieler erweitert werden. Die Methode `torschuetzenkoenig` soll den Namen eines Spielers aus dem Team zurückgeben, der die meisten Tore für die Mannschaft geschossen hat. Ist das für mehrere Spieler der Fall, dann kann der Name eines beliebigen solchen Spielers zurückgegeben werden. Ist noch kein Spieler im Team, dann soll der String `"Kein Spieler vorhanden"` zurückgegeben werden.

Lösungsvorschlag

```java
public class Mannschaft {
  @SuppressWarnings("unused")
  private String land;
  private int anzahlSpieler;
  private Spieler[] team;

  public Mannschaft(String land) {
    this.land = land;
    anzahlSpieler = 0;
    team = new Spieler[20];
  }

  public void hinzufuegen(Spieler s) {
    team[anzahlSpieler] = s;
    anzahlSpieler++;
  }

  public String torschuetzenkoenig() {
    if (anzahlSpieler == 0) {
      return "Kein Spieler vorhanden";
    }
    Spieler koenig = team[0];
    for (int i = 0; i < anzahlSpieler; i++) {
      Spieler spieler = team[i];
      if (spieler.getTore() > koenig.getTore()) {
        koenig = spieler;
      }
    }
    return koenig.getName();
  }
```

```
}
```

main-Methode
Zustandsdiagramm zeichnen

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2018/fruehjahr/verein/Mannschaft.java`

(c) Schreiben Sie den Rumpf einer `main`-Methode, so dass nach Ausführung der Methode eine deutsche Mannschaft existiert mit zwei Spielern Namens `"Hugo Meier"` und `"Frank Huber"`. Beide Spieler sollen zum selben Verein `"FC Staatsexamen"` gehören. `"Hugo Meier"` soll nach Aufnahme in die deutsche Mannschaft genau ein Tor geschossen haben, während `"Frank Huber"` noch kein Tor erzielt hat. (Wir abstrahieren hier von der Realität, in der ein 2-er Team noch gar nicht spielbereit ist.)

Lösungsvorschlag

```java
public class Verein {
  private String name;

  public Verein(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }

  public static void main(String[] args) {
    Verein verein1 = new Verein("FC Staatsexamen");
    Spieler spieler1 = new Spieler("Hugo Meier", verein1);
    Spieler spieler2 = new Spieler("Frank Huber", verein1);
    Mannschaft deutscheMannschaft = new Mannschaft("Deutschland");
    deutscheMannschaft.hinzufuegen(spieler1);
    deutscheMannschaft.hinzufuegen(spieler2);
    spieler1.schiesseTor();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2018/fruehjahr/verein/Verein.java`

46116 / 2018 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 3

Eine Digitaluhr kann alternativ entweder die Zeit (Stunden und Minuten) oder das Datum (Tag, Monat und Jahr) anzeigen. Zu Beginn zeigt die Uhr die Zeit an. Sie besitzt drei Druckknöpfe **A**, **B** und **C**. Mit Knopf **A** kann zwischen Zeit- und Datumsanzeige hin und her gewechselt werden.

Wird die Zeit angezeigt, dann kann mit Knopf **B** der Reihe nach erst in einen Stundenmodus, dann in einen Minutenmodus und schließlich zurück zur Zeitanzeige gewechselt werden. Im Stundenmodus blinkt die Stundenanzeige. Mit Drücken des Knopfes **C** können dann die Stunden schrittweise inkrementiert werden. Im Minutenmodus blinkt die Minutenanzeige und es können mit Hilfe des Knopfes **C** die Minuten schrittweise inkrementiert werden.

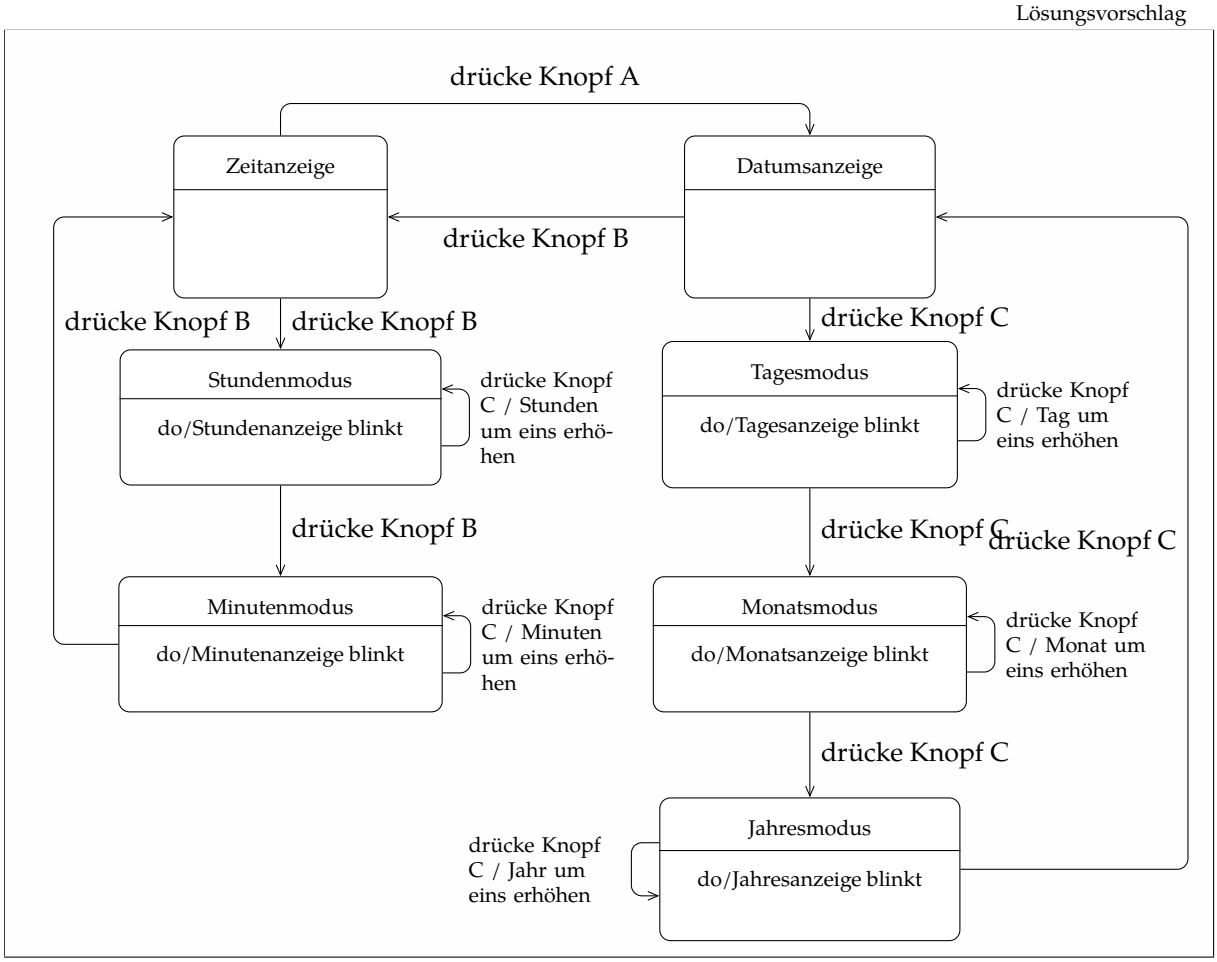
Die Datumsfunktionen sind analog. Wird das Datum angezeigt, dann kann mit Knopf **B** der Reihe nach in einen Tagesmodus, Monatsmodus, Jahresmodus und schließlich zurück zur Datumsanzeige gewechselt werden. Im Tagesmodus blinkt die Tagesanzeige. Mit Drücken des Knopfes **C** können dann die Tage schrittweise inkrementiert werden. Analog blinken mit Eintritt in den entsprechenden Einstellmodus der Monat oder das Jahr, die dann mit Knopf **C** schrittweise inkrementiert werden können.

Wenn sich die Uhr in einem Einstellmodus befindet, hat das Betätigen des Knopfes **A** keine Wirkung. Ebenso wirkungslos ist Knopf **C**, wenn gerade Zeit oder Datum angezeigt wird.

Beschreiben Sie das Verhalten der Digitaluhr durch ein UML-Zustandsdiagramm. Dabei muss - gemäß der UML-Notation - unterscheidbar sein, was Ereignisse und was Aktionen sind. Deren Bedeutung soll durch die Verwendung von sprechenden Namen klar sein. Für die Inkrementierung von Stunden, Minuten, Tagen etc. brauchen keine konkreten Berechnungen angegeben werden. Der kontinuierliche Zeitfortschritt des Uhrwerks ist nicht zu modellieren.

Zustände sind, wie in der UML üblich, durch abgerundete Rechtecke darzustellen. Sie können unterteilt werden in eine obere und eine untere Hälfte, wobei der Name des Zustands in den oberen Teil und eine in dem Zustand auszuführende Aktivität in den unteren Teil einzutragen ist.

Lösungsvorschlag



66112 / 2002 / Herbst / Thema 1 / Aufgabe 4

Vererbung
Abstrakte Klasse
Implementierung in Java
Klassendiagramm

In einer Anforderungsanalyse für ein Banksystem wird der folgende Sachverhalt beschrieben:

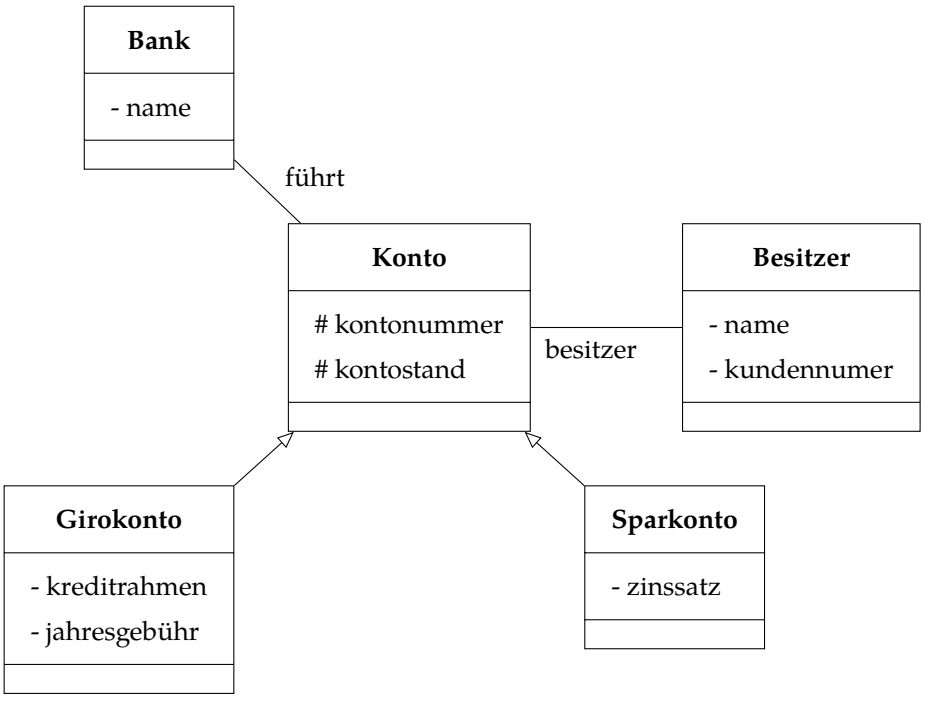
Eine Bank hat einen Namen und sie führt Konten. Jedes Konto hat eine Kontonummer, einen Kontostand und einen Besitzer. Der Besitzer hat einen Namen und eine Kundennummer. Ein Konto ist entweder ein Sparkonto oder ein Girokonto. Ein Sparkonto hat einen Zinssatz, ein Girokonto hat einen Kreditrahmen und eine Jahresgebühr.

(a) Deklarieren Sie geeignete Klassen in Java [oder in C++], die die oben beschriebenen Anforderungen widerspiegeln! Nutzen Sie dabei das Vererbungskonzept aus, wo es sinnvoll ist! Gibt es Klassen, die als abstrakt zu verstehen sind?

[Hinweis: Geben Sie sowohl ein Klassendiagramm als auch den Quellcode für die beteiligten Klassen inkl. Attributen, ohne Methoden an! Achtung: in Teilaufgabe b) werden anschließend die benötigten Konstruktoren verlangt; Um die verschiedenen Konten in einer Bank zu verwalten, eignet sich das Array NICHT als Datenstruktur. Informieren Sie sich über die Datenstruktur „ArrayList“ und verwenden Sie diese.]

Lösungsvorschlag

Die Klasse Konto ist abstrakt, da ein Konto immer entweder ein Spar- *oder* ein Girokonto ist. Ein Objekt der Klasse Konto ist deshalb nicht sinnvoll.



(b) Geben Sie für alle nicht abstrakten Klassen benutzerdefinierte Konstruktoren an mit Parametern zur Initialisierung der folgenden Werte: der Name einer Bank, die Kontonummer, der Kontostand, der Besitzer und der Zinssatz (bzw. Kreditrahmen und Jahresgebühr) eines Sparkontos (bzw. Girokontos), der Name und die Kundennummer eines Kontobesitzers.

Ergänzen Sie die Klassen um Methoden für die folgenden Aufgaben! Nutzen Sie wann immer möglich das Vererbungskonzept aus und verwenden Sie ggf. abstrakte (bzw. virtuelle) Methoden!

[Achtung: Ergänzen Sie ggf. alle benötigten Getter und Setter in den beteiligten Klassen!]

Lösungsvorschlag

> Konstruktoren ergänzen:
>
> Bemerkung: Auch eine abstrakte Klasse kann einen Konstruktor besitzen, dieser kann nur nicht ausgeführt werden. In den abgeleiteten Klassen kann dieser Super-Konstruktor aber verwendet werden.

(c) Auf ein Konto soll ein Betrag eingezahlt und ein Betrag abgehoben werden können. Soll von einem Sparkonto ein Betrag abgehoben werden, dann darf der Kontostand nicht negativ werden. Bei einer Abhebung von einem Girokonto darf der Kreditrahmen nicht überzogen werden.

Lösungsvorschlag

> Bemerkung: Die Methode `einzahlen` ist in der Klasse `Konto` implementiert, da sie sich für Spar- und Girokonten nicht unterscheidet im Gegensatz zur Methode `abheben`, die in beiden Klassen unterschiedlich implementiert ist. [Sie wird als abstrakte Methode in der Klasse Konto angegeben, um ihre Implementierung (Überschreiben) zu gewährleisten.] Kreditrahmen wird als negativer Wert gespeichert. Die Methoden zum Abheben liefern zusätzlich zur Änderung des Kontostandes eine Rückmeldung bezüglich Erfolg oder Misserfolg der Abbuchung.

(d) Ein Konto kann eine Jahresabrechnung durchführen. Bei der Jahresabrechnung eines Sparkontos wird der Zinsertrag gutgeschrieben, bei der Jahresabrechnung eines Girokontos wird die Jahresgebühr abgezogen (auch wenn dadurch der Kreditrahmen überzogen wird).

Lösungsvorschlag

> Anmerkung: Im Folgenden nur noch Angabe der gesuchten Methoden, alle vorherigen Implementierungen (Attribute, Konstruktoren, Methoden, s. o.) wurden nicht nochmals aufgeführt.

(e) Eine Bank kann einen Jahresabschluss durchführen. Dieser bewirkt, dass für jedes Konto der Bank eine Jahresabrechnung durchgeführt wird.

(f) Eine Bank kann ein Sparkonto eröffnen. Die Methode soll die folgenden fünf Parameter haben: den Namen und die Kundennummer des Kontobesitzers, die Kontonummer, den (anfänglichen) Kontostand und den Zinssatz des Sparkontos. Alle Parameter sind als String-Objekte oder als Werte eines Grunddatentyps zu übergeben! Das Sparkonto muss nach seiner Eröffnung in den Kontenbestand der Bank aufgenommen sein.

[Hinweis: Falls der Kunde schon mit einem Konto in der Bank geführt ist, können die Werte für das `Besitzer`-Objekt übernommen werden. Schreiben Sie daher eine Hilfsmethode `Besitzer schonVorhanden(String name, int kunr)`, die prüft, ob der Kunde mit `name` und `kunr` schon in der Liste vorhanden ist und diesen bzw. andernfalls einen neu erzeugten Besitzer zurückgibt.]

[Sinnvolle/notwendige Methoden der Klasse ArrayList für diese Aufgabe:
`public int size()`: Returns the number of elements in this list.
`public boolean isEmpty()`: Returns true if this list contains no elements.
`public E get(int index)`: Returns the element at the specified position in this list.
`public boolean add(E e)`: Appends the specified element to the end of this list.
(siehe auch Java-Api: https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html)]

Lösungsvorschlag

```
import java.util.ArrayList;

public class Bank {
  @SuppressWarnings("unused")
  private String name;
  private ArrayList<Konto> konten;

  public Bank(String name) {
    this.name = name;
    konten = new ArrayList<Konto>();
  }

  public void rechneAb() {
    for (int i = 0; i <= konten.size(); i++) {
      konten.get(i).rechneAb();
    }
  }

  public void legeAn(String name, int kundenNummer, int kontoNummer, float
  ↪ kontoStand, float zinssatz) {
    Besitzer besitzer = schonVorhanden(name, kundenNummer);
    konten.add(new Sparkonto(kontoNummer, kontoStand, besitzer, zinssatz));
  }

  public Besitzer schonVorhanden(String name, int kundenNummer) {
    if (!konten.isEmpty()) {
      for (int i = 0; i < konten.size(); i++) {
        if (konten.get(i).besitzer.gibKundenNummer() == kundenNummer) {
          return konten.get(i).gibBesitzer();
        }
      }
    }
    return new Besitzer(name, kundenNummer);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Bank.java`

```
public abstract class Konto {
```

```
  protected int kontoNummer;
  protected float kontoStand;
  protected Besitzer besitzer;

  public Konto(int kontoNummer, float kontoStand, Besitzer besitzer) {
    this.kontoNummer = kontoNummer;
    this.kontoStand = kontoStand;
    this.besitzer = besitzer;
  }

  public void zahleEin(float betrag) {
    kontoStand += betrag;
  }

  public Besitzer gibBesitzer() {
    return besitzer;
  }

  public abstract boolean hebeAb(float betrag);

  public abstract void rechneAb();

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Konto.java`

```
public class Sparkonto extends Konto {
  private float zinssatz;

  public Sparkonto(int kontoNummer, float kontoStand, Besitzer besitzer,
  ↪  float zinssatz) {
    super(kontoNummer, kontoStand, besitzer);
    this.zinssatz = zinssatz;
  }

  public boolean hebeAb(float betrag) {
    if (kontoStand - betrag >= 0) {
      kontoStand -= betrag;
      return true;
    } else {
      return false;
    }
  }

  public void rechneAb() {
    kontoStand += kontoStand * (1 + zinssatz);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Sparkonto.java`

```
public class Girokonto extends Konto {
  private float kreditRahmen;
  private float jahresGebühr;
```

Klassendiagramm

```
  public Girokonto(int kontoNummer, float kontoStand, Besitzer besitzer,
↪   float kreditRahmen, float jahresGebühr) {
    super(kontoNummer, kontoStand, besitzer);
    this.kreditRahmen = kreditRahmen;
    this.jahresGebühr = jahresGebühr;
  }

  public boolean hebeAb(float betrag) {
    if (kontoStand - betrag >= kreditRahmen) {
      kontoStand -= betrag;
      return true;
    } else {
      return false;
    }
  }

  public void rechneAb() {
    kontoStand -= jahresGebühr;
  }
}
```

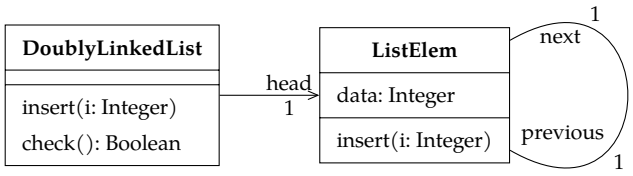
Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Girokonto.java`

```
@SuppressWarnings("unused")
public class Besitzer {
  private String name;
  private int kundenNummer;
  private Konto hatKonto;

  public Besitzer(String name, int kundenNummer) {
    this.name = name;
    this.kundenNummer = kundenNummer;
  }

  public int gibKundenNummer() {
    return kundenNummer;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Besitzer.java`

66112 / 2005 / Frühjahr / Thema 1 / Aufgabe 1

Betrachten Sie folgendes Klassendiagramm, das doppelt-verkettete Listen spezifiziert. Die Assoziation `head` zeigt auf das erste Element der Liste. Die Assoziationen `previous` und `next` zeigen auf das vorherige bzw. folgende Element.



Implementieren Sie die doppelt-verketteten Listen in einer geeigneten objektorientierten Sprache (z. B. Java oder C++), das heißt:

(a) Implementieren Sie die Klasse `ListElem`. Die Methode `insert` ordnet eine ganze Zahl `i` in eine aufsteigend geordnete doppelt-verkettete Liste `l` an die korrekte Stelle ein. Sei z. B. das Objekt `l` eine Repräsentation der Liste `[0, 2, 2, 6, 8]` dann liefert `l.insert(3)` eine Repräsentation der Liste `[0, 2, 2, 3, 6, 8]`.

Lösungsvorschlag

```
public class ListElem {
  private int data;
  private ListElem previous;
  private ListElem next;

  public ListElem(int i) {
    data = i;
  }

  public ListElem() {
  }

  public void insert(int i) {
    ListElem newElement = new ListElem(i);
    if (i <= data) {
      if (previous != null) {
        newElement.next = this;
        newElement.previous = previous;
        previous.next = newElement;
        previous = newElement;
      } else {
        newElement.next = this;
        previous = newElement;
      }
    } else {
      if (next != null) {
        next.insert(i);
      } else {
        newElement.previous = this;
        next = newElement;
      }
    }
  }

  public ListElem getPrevious() {
```

```
    return previous;
  }

  public ListElem getNext() {
    return next;
  }

  public int getData() {
    return data;
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66112/jahr_2005/fruehjahr/ListElem.java

(b) Implementieren Sie die Klasse `DoublyLinkedList`, wobei die Methode `insert` eine Zahl `i` in eine aufsteigend geordnete Liste einordnet. Die Methode `check` überprüft, ob eine Liste korrekt verkettet ist, ðob für jedes ListElem-Objekt `o`, das über den `head` der Liste erreichbar ist, der Vorgänger des Nachfolgers von `o` gleich `o` ist.

Lösungsvorschlag

```
public class DoublyLinkedList {
  private ListElem head;

  public DoublyLinkedList() {
  }

  public void insert(int i) {
    if (head != null) {
      // Immer einen neue Zahl einfügen, nicht nur wenn die Zahl kleiner ist als
↪   head.
      head.insert(i);
      // Es muss kleiner gleich heißen, sonst können mehrer gleiche Zahlen am
↪   Anfang
      // nicht eingefügt werden.
      if (i <= head.getData()) {
        head = head.getPrevious();
      }
    } else {
      head = new ListElem(i);
    }
  }

  public boolean check() {
    ListElem current = head;
    while (current.getNext() != null) {
      if (current.getNext().getPrevious() != current) {
        return false;
      } else {
        current = current.getNext();
      }
    }
    return true;
```

```
  }

  public ListElem getHead() {
    return head;
  }

  public static void main(String[] args) {
    DoublyLinkedList list = new DoublyLinkedList();
    // int[] numbers = new int[] { 1 };
    // int[] numbers = new int[] { 1, 1, 1, 1, };
    // int[] numbers = new int[] { 1, 1, 1, 2, };
    // int[] numbers = new int[] { 2, 1, 1, 1, };
    // int[] numbers = new int[] { 2, 1 };
    int[] numbers = new int[] { 0, 2, 2, 6, 8, 4 };
    for (int number : numbers) {
      list.insert(number);
    }
    list.insert(3);

    ListElem current = list.getHead();
    while (current.getNext() != null) {
      System.out.println(current.getData());
      current = current.getNext();
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2005/fruehjahr/DoublyLinkedList.java`

Modell-Präsentation-Steuerung (Model-View-Controller)

66116 / 2014 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 2

Sie sollen das Design für ein einfaches Wahlsystem entwerfen. Das System soll dabei die Verteilung der Stimmen auf die einzelnen Parteien ermöglichen. Zusätzlich soll es verschiedene Darstellungen dieser Daten erlauben: Eine Tabelle, in der die Daten gelesen und auch eingegeben werden können, und ein Diagramm als alternative Darstellung der Informationen. Das System soll mit dem *Model-View-Controller* Muster modelliert werden.

(a) Beschreiben Sie das *Model-View-Controller* Muster:

(i) Beschreiben Sie das Problem, welches das Muster adressiert.

Lösungsvorschlag

> Das MVC-Muster wird verwendet, um spätere Änderungen bzw. Erweiterungen zu vereinfachen. Dies unterstützt somit auch die Wiederverwendbarkeit.

(ii) Beschreiben Sie die Aufgaben der Komponenten, die im Muster verwendet werden.

Lösungsvorschlag Implementierung in Java
Zustandsdiagramm zeichnen

> Im Modell werden die Daten verwaltet, die View ist für die Darstellung der daten sowie die Benutzerinteraktion zuständig und der Controller übernimmt die Steue- rung zwischen View und Modell.
> Das MVC-Muster ist aus den drei Entwurfsmustern Beobachter, Kompositum und Strategie zusammengesetzt.

(b) Modellieren Sie das System unter Anwendung des Musters:

(i) Entwerfen Sie ein UML Klassendiagramm.

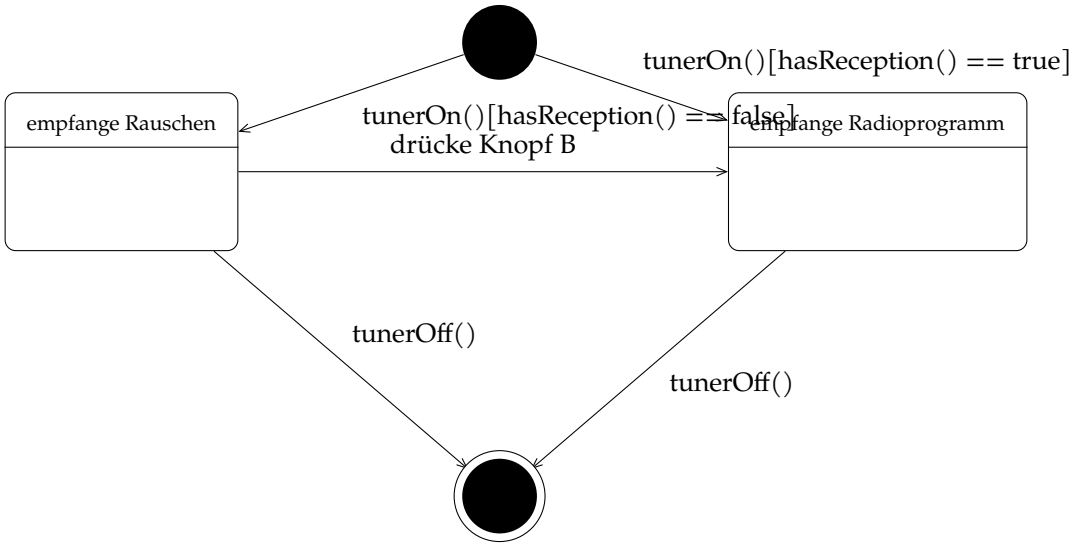
(ii) Implementieren Sie eine setup-Methode, die das Objektmodell erstellt.

66116 / 2015 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 2

Erstellen Sie ein UML-Zustandsdiagramm für einen Radiotuner. Mit dem Radiotuner können Sie ein Radioprogramm auf der Frequenz f empfangen. Beim Einschalten wird f auf 87,5 MHz gesetzt und der Tuner empfängt. Sie können nun die Frequenz in 0,5 MHz Schritten erhöhen oder senken. Bitte beachten Sie, dass das Frequenzband für den Radioempfang von 87,5 MHz bis 108 MHz reicht. Sobald f 87,5 MHz unter- bzw. 108 MHz überschreitet, soll f auf 108 MHz bzw. 87,5 MHz gesetzt werden. Weiterhin kann ein Suchmodus gestartet werden, der automatisch die Frequenz erhöht, bis ein Sender empfangen wird. Wird während des Suchmodus die Frequenz verändert oder erneut Suchen ausgeführt, dann wird die Suche beendet (und, je nach Knopf, ggf. noch die Frequenz um 0,5 MHz verändert).
Die Klasse RadioTuner besitzt folgende Methoden:

- tunerOn()
- tunerOff()
- increaseFrequency()
- decreaseFrequency()
- seek()
- hasReception()

Hinweis: Die Hilfsmethode hasReception() liefert true zurück, genau dann wenn ein Sender empfangen wird.



66116 / 2015 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 3

In dieser Aufgabe implementieren Sie ein konzeptionelles Datenmodell für eine Firma, die Personendaten von Angestellten und Kunden verwalten möchte. Gegeben seien dazu folgende Aussagen:

- Eine *Person* hat einen *Namen* und ein *Geschlecht* (männlich oder weiblich).
- Ein *Angestellter* ist eine *Person*, zu der zusätzlich das monatliche *Gehalt* gespeichert wird.
- Ein *Kunde* ist eine *Person*, zu der zusätzlich eine *Kundennummer* hinterlegt wird.

(a) Geben Sie in einer objektorientierten Programmiersprache Ihrer Wahl (geben Sie diese an) eine Implementierung des aus den obigen Aussagen resultierenden konzeptionellen Datenmodells in Form von **Klassen** und **Interfaces** an. Gehen Sie dabei wie folgt vor:

- Schreiben Sie ein Interface `Person` sowie zwei davon erbende Interfaces `Angestellter` und `Kunde`. Die Interfaces sollen jeweils lesende Zugriffsmethoden (Getter) die entsprechenden Attribute (Name, Geschlecht, Gehalt, Kundennummer) deklarieren.

Lösungsvorschlag

```java
public interface Person {
  String getName();

  char getGeschlecht();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/Person.java`

```java
public interface Angestellter extends Person {
  int getGehalt();
}
```

Abstrakte Klasse

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/Angestellter.java`

```
public interface Kunde extends Person {
  int getKundennummer();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/Kunde.java`

- Schreiben Sie eine abstrakte Klasse `PersonImpl`, die das Interface `Person` implementiert. Für jedes Attribut soll ein Objektfeld angelegt werden. Außerdem soll ein Konstruktor definiert werden, der alle Objektfelder initialisiert.

Lösungsvorschlag

```
public abstract class PersonImpl implements Person {
  protected String name;
  protected char geschlecht;

  public PersonImpl(String name, char geschlecht) {
    this.name = name;
    this.geschlecht = geschlecht;
  }

  public String getName() {
    return name;
  }

  public char getGeschlecht() {
    return geschlecht;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/PersonImpl.java`

- Schreiben Sie zwei Klassen `AngestellterImpl` und `KundeImpl`, die von `PersonImpl` erben und die jeweils dazugehörigen Interfaces implementieren. Es sollen wiederum Konstruktoren definiert werden, die alle Objektfelder initialisieren und dabei auf den Konstruktor der Basisklasse `PersonImpl` Bezug nehmen.

Lösungsvorschlag

```
public class AngestellterImpl extends PersonImpl implements Angestellter
↪ {
  protected int gehalt;

  public AngestellterImpl(String name, char geschlecht, int gehalt) {
    super(name, geschlecht);
    this.gehalt = gehalt;
  }

  public int getGehalt() {
    return gehalt;
  }

}
```

Adapter

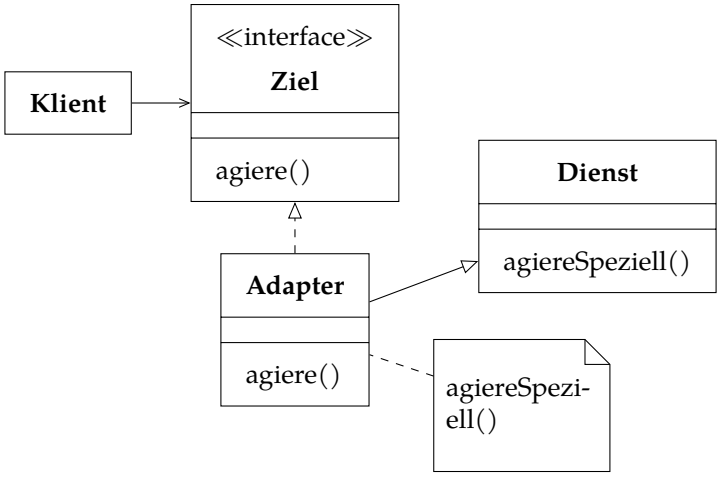
Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/AngestellterImpl.java

```
public class KundeImpl extends PersonImpl implements Kunde {
  protected int kundennummer;

  public KundeImpl(String name, char geschlecht, int kundennummer) {
    super(name, geschlecht);
    this.kundennummer = kundennummer;
  }

  public int getKundennummer() {
    return kundennummer;
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/KundeImpl.java

(b) Verwenden Sie das Entwurfsmuster **Adapter**, um zu ermöglichen, dass vorhandene Angestellte die Rolle eines Kunden einnehmen können. Der Adapter soll eine zusätzliche Klasse sein, die das Kunden-Interface implementiert. Wenn möglich, sollen Methodenaufrufe an den adaptierten Angestellten delegiert werden. Möglicherweise müssen Sie neue Objektfelder einführen.

Exkurs: Entwurfsmuster „Adapter“



Ziel (Target) Das Ziel definiert die Schnittstelle, die der Klient nutzen kann.

Klient (Client) Der Klient nutzt Dienste über inkompatible Schnittstellen und greift dabei auf adaptierte Schnittstellen zurück.

Dienst (Adaptee) Der Dienst bietet wiederzuverwendende Dienstleistungen mit fest definierter Schnittstelle an.

Adapter Der Adapter adaptiert die Schnittstelle des Dienstes auf die Schnittstelle zum Klienten.

```
/**
 * GoF: Adaptee
 */
public class Dienst {
```

```
  public void agiereSpeziell() {
      System.out.println("Agiere speziell!");
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/adapter/allgemein/Dienst.java`

```
public interface Ziel {
  public void agiere();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/adapter/allgemein/Ziel.java`

```
public class Adapter extends Dienst implements Ziel {

  @Override
  public void agiere() {
      System.out.print("agiere: ");
      agiereSpeziell();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/adapter/allgemein/Adapter.java`

```
public class Klient {

  public static void main(String[] args) {
    new Adapter().agiere();
    // agiere: Agiere speziell!
  }
}
```

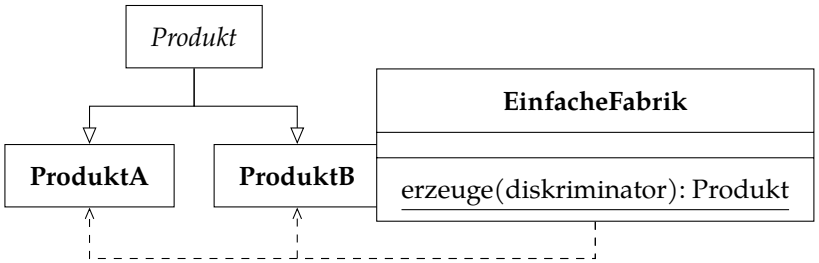
Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/entwurfsmuster/adapter/allgemein/Klient.java`

(c) Verwenden Sie das Entwurfsmuster **Simple Factory**, um die Erzeugung von Angestellten, Kunden, sowie Adapter-Instanzen aus Aufgabe (b) zu vereinheitlichen. Die entsprechende Erzeugungs-Methode soll neben einem Typ-Diskriminator (z. B. einem Aufzählungstypen oder mehreren booleschen Werten) alle Parameter übergeben bekommen, die für den Konstruktor irgendeiner Implementierungsklasse des Interface `Person` notwendig sind.

Hinweis: Um eine Adapter-Instanz zu erzeugen, müssen Sie möglicherweise zwei Konstruktoren aufrufen.

Exkurs: Entwurfsmuster „Einfache Fabrik“

Klassendiagramm



EinfacheFabrik Eine Klasse mit einer Erzeugunsmethode, die über eine größere Bedingung verschiedene Objekt instanziert.

Produkt Eine abstrakte Klasse, die von den konkreten Produkten geerbt wird.

KonkretesProdukt Ein konkretes Produkt, das von der einfachen Fabrik erzeugt wird.

66116 / 2015 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 2

Leider ist das Klassendiagramm der folgenden Klassen verloren gegangen. Führen Sie ein Reverse Engineering durch und erstellen Sie aus dem Quellcode ein vollständiges UML-Klassendiagramm inklusive aller Klassen, Schnittstellen, Attribute, Methoden, Konstruktoren, Sichtbarkeiten, Assoziationen, Rollennamen, Multiplizitäten, Navigationspfeilen und evtl. Stereotypen. Der Quellcode innerhalb von Methoden und Konstruktoren soll nicht übertragen werden, wohl aber die Methodensignaturen. Assoziationsnamen und deren Leserichtung lassen sich aus dem Quellcode nur schwer erahnen und sollen deshalb ebenfalls weggelassen werden.

```
public abstract class Display implements PixelPainter {
  protected HardwareMatrix hardwareMatrix;
  protected int lastPaintedX;
  protected int lastPaintedY;

  public Display(HardwareMatrix hardwareMatrix) {
    this.hardwareMatrix = hardwareMatrix;

  }

  public int getWidth() {
    return hardwareMatrix.getWidth() / getWidthFactor();
  }

  public int getHeight() {
    return hardwareMatrix.getHeight() / getHeightFactor();
  }

  public void clear() {
    // some longer code
  }

  protected abstract int getWidthFactor();

  protected abstract int getHeightFactor();
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/reverse/Display.java

```
import java.awt.Color;

public interface PixelPainter {
  void set(int x, int y, Color color);

  int getHeight();

  int getWidth();
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/reverse/PixelPainter.java

```
public interface HardwareMatrix {
  void set(int x, int y, int v);

  int getWidth();

  int getHeight();
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/reverse/HardwareMatrix.java

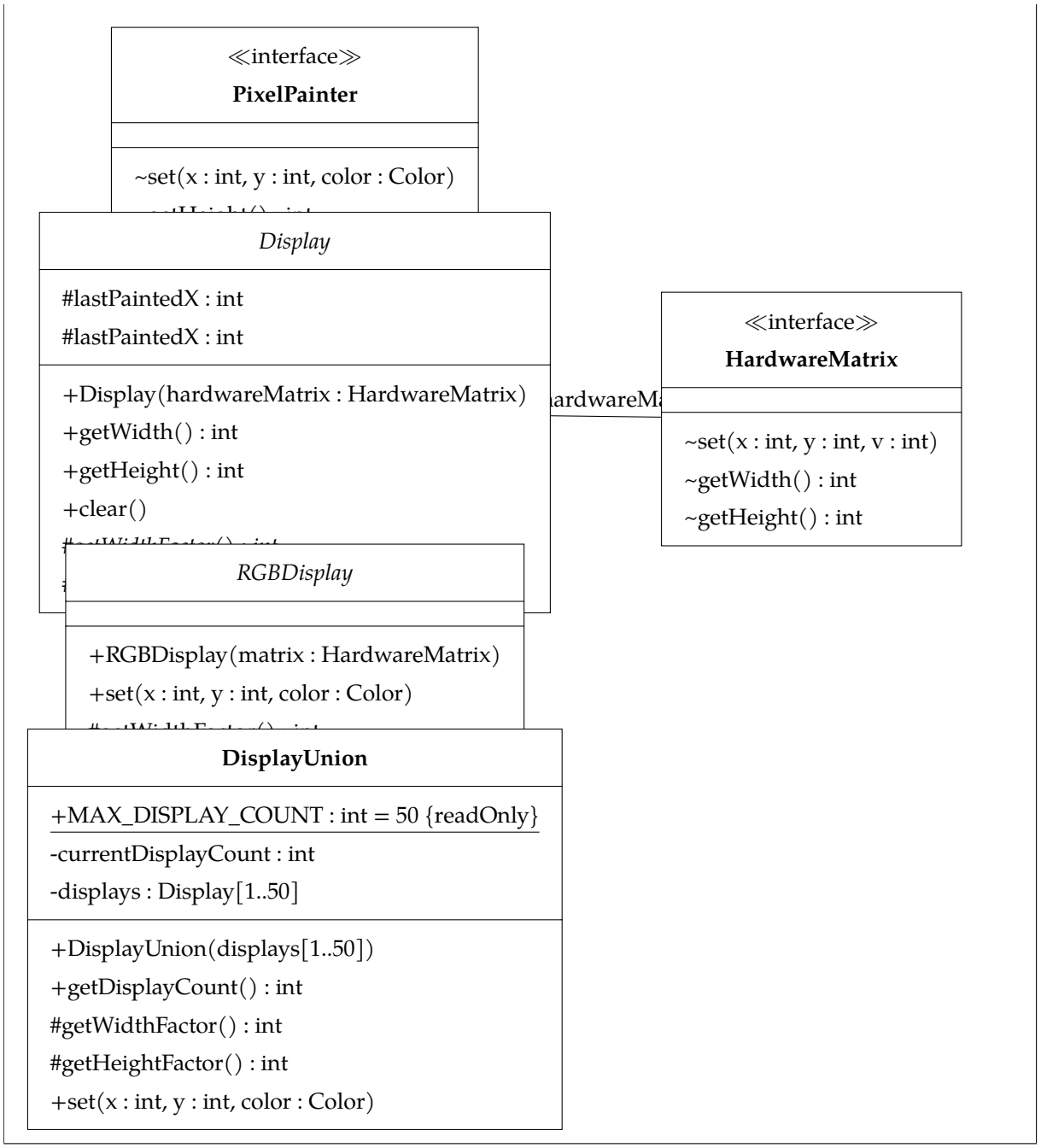
```
import java.awt.Color;

@SuppressWarnings({ "unused" })
public class DisplayUnion extends RGBDisplay {
  public static final int MAX_DISPLAY_COUNT = 50;

  private int currentDisplayCount;

  private Display[] displays;

  public DisplayUnion(Display[] displays) {
    super(null);
  }

  public int getDisplayCount() {
    return 0;
  }

  protected int getWidthFactor() {
    return 1;
  }

  protected int getHeightFactor() {
    return 1;
  }

  public void set(int x, int y, Color color) {
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/reverse/DisplayUnion.java

```java
import java.awt.Color;

public class RGBDisplay extends Display {
  public RGBDisplay(HardwareMatrix matrix) {
    super(matrix);
  }

  public void set(int x, int y, Color color) {
  }

  protected int getWidthFactor() {
    return 3;
  }

  protected int getHeightFactor() {
    return 1;
  }
}
```

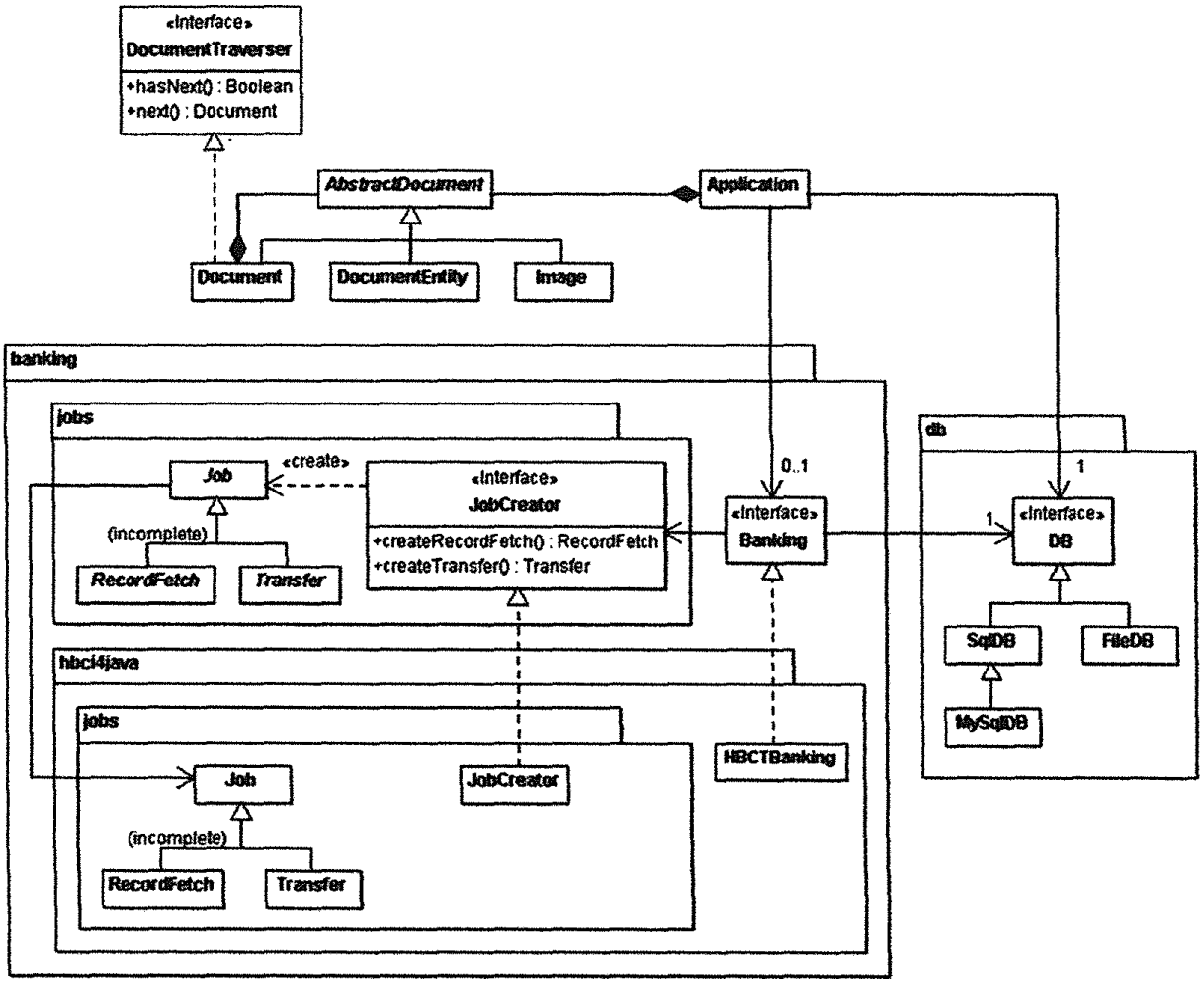
Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/reverse/RGBDisplay.java`

Lösungsvorschlag


≪interface≫
PixelPainter
~set(x : int, y : int, color : Color)
Display
#lastPaintedX : int
#lastPaintedX : int
+Display(hardwareMatrix : HardwareMatrix)
+getWidth() : int
+getHeight() : int
+clear()
≪interface≫
HardwareMatrix
~set(x : int, y : int, v : int)
~getWidth() : int
~getHeight() : int
RGBDisplay
+RGBDisplay(matrix : HardwareMatrix)
+set(x : int, y : int, color : Color)
DisplayUnion
+MAX_DISPLAY_COUNT : int = 50 {readOnly}
-currentDisplayCount : int
-displays : Display[1..50]
+DisplayUnion(displays[1..50])
+getDisplayCount() : int
#getWidthFactor() : int
#getHeightFactor() : int
+set(x : int, y : int, color : Color)

66116 / 2016 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 2

Klassendiagramm
Abstrakte Fabrik (Abstract Factory)
Wiederholer (Iterator)
Adapter
Kompositum (Composite)



(a) Kennzeichnen Sie im folgenden Klassendiagramm die Entwurfsmuster *„Abstrakte Fabrik“*, *„Iterator“*, *„Adapter “* und *„Kompositum“*. Geben Sie die jeweils beteiligten Klassen und deren Zuständigkeit im entsprechenden Muster an.

Lösungsvorschlag

Iterator

DocumentTraverser (interface) Schnittstelle zur Traversierung und zum Zugriff auf Dokumente

Document implementiert die Schnittstelle

Kompositum

AbstractDocument abstrakte Basisklasse, die gemeinsames Verhalten der beteiligten Klassen definiert

Document enthält wiederum weitere Documente bzw. DocumentEntities und Images

DocumentEntity, Image primitive Unterklassen, besitzen keine Kindobjekte

Adapter (Objektadapter)

Banking (interface) vom Client (hier Application) verwendete Schnittstelle

HBCTBanking passt Schnittstelle der unpassenden Klasse an Zielschnittstelle (Banking) an

DB (interface) anzupassende Schnittstelle

abstrakte Fabrik

JobCreator (interface) abstrakte Fabrik

Job (abstrakt) mit Unterklassen RecordFetch und Transfer abstraktes Produkt

Job (konkret) mit Unterklassen konkretes Produkt

(b) (i) Beschreiben Sie die Funktionsweise der folgenden Entwurfsmuster und geben Sie ein passendes UML-Diagramm an.

- Dekorierer
- Klassenadapter
- Objektadapter

(ii) Erklären Sie mit maximal zwei Sätzen den Unterschied zwischen Klassenadapter und Objektadapter.

(c) Implementieren Sie einen Stapel in der Programmiersprache Java. Nutzen Sie dazu ein Array mit fester Größe. Auf eine Überlaufprüfung darf verzichtet werden. Implementieren Sie in der Klasse das Iterator Entwurfsmuster, um auf die Inhalte zuzugreifen, sowie eine Funktion zum Hinzufügen von Elementen. Als Typ für den Stapel kann zur Vereinfachung ein Integertyp verwendet werden.

66116 / 2016 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 2

(a) Gegeben sei folgende natürlichsprachliche Spezifikation eines PKI-Systems:

Damit der Schüler seinem Lehrer die Hausaufgaben verschlüsselt per E-Mail übermitteln kann, bedarf es einer entsprechenden Infrastruktur. Nachdem beide Teilnehmer die notwendige Software installiert haben, erstellt der Lehrer zunächst ein sogenanntes Schlüsselpaar, bestehend aus einem „Öffentlichen“ (ÖS) und einem zugehörigen „privaten“ Schlüssel (PS). Anschließend veröffentlicht der Lehrer seinen ÖS durch Hochladen auf einen sogenannten Keyserver (Schlüsselverzeichnisdienst). Damit steht er jedem Schüler zur Verfügung, so dass dieser den ÖS jederzeit vom Keyserver herunterladen kann. Alternativ kann der Lehrer seinen ÖS auch direkt (z. B. per E-Mail oder USB-Stick) an den Schüler übermitteln.

Der Schüler kann nun seine Nachricht (z. B. seine Lösung) mit dem ÖS des Lehrers verschlüsseln und mit einer E-Mail versenden. Empfängt der Lehrer eine solche E-Mail, so kann er (und nur er) mit seinem PS die Nachricht wieder entschlüsseln. Umgekehrt kann der Lehrer mit seinem PS beliebige Informationen (z. B. die Note) „digital signieren“. Diese unterschriebenen Daten übermittelt der Lehrer dann zusammen mit der digitalen

Signatur per E-Mail an den Schüler. Der Schüler kann mit dem ÖS des Lehrers prüfen, ob die übermittelte Nachricht unverändert und tatsächlich vom unterschreibenden Lehrer stammt.

Modellieren Sie die in der Spezifikation beschriebenen Anwendungsfälle zusammen mit den jeweils beteiligten Aktoren in einem Use-Case-Diagramm. Betrachten Sie den Keyserver zunächst ebenfalls als Aktor, welcher gegenüber PKI als „externer Vermittler“ auftritt.

(b) Erstellen Sie ein geeignetes Klassendiagramm für das obige PKI. Berücksichtigen Sie zusätzlich zur verbalen Spezifikation noch folgende Präzisierungen:

(i) Die Schlüssel werden mit einer E-Mail-Adresse „benannt“, damit der Schüler den richtigen Schlüssel abrufen kann.

(ii) Es gibt genau einen Keyserver; dieser verwaltet aber beliebig viele Schlüssel. Er bietet die entsprechenden Dienste zum Veröffentlichen bzw. Abfragen von Schlüsseln an.

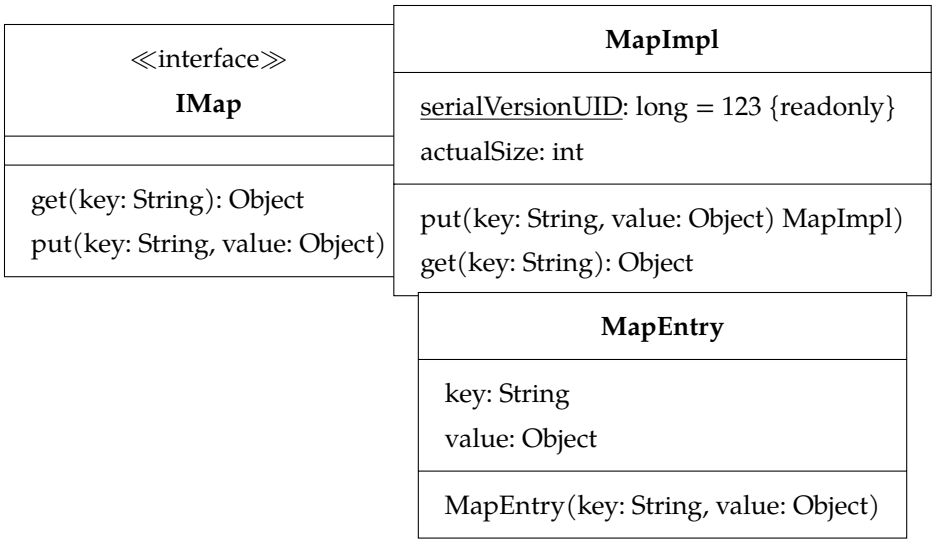
(iii) Jeder Lehrer hat höchstens ein Schlüsselpaar, aber jeder Schlüssel gehört genau einem Lehrer. Der Schüler hingegen kommuniziert mit mehreren Lehrern und kennt daher mehrere E-Mail-Adressen.

(iv) Eine Nachricht kann (muss aber nicht) eine Signatur oder einen ÖS als „Anhang“ zusätzlich zum eigentlichen Inhalt (zur Vereinfachung: String) mit sich führen.

(v) Für das Signieren bzw. Entschlüsseln ist der PS (das zugehörige Objekt selbst) zuständig. Dafür bekommt er den Inhalt der Nachricht und gibt entsprechend eine Signatur bzw. den entschlüsselten Inhalt zurück.

(vi) Für das Prüfen der Signatur und das Verschlüsseln ist der ÖS zuständig. Dazu bekommen die Methoden je nach Bedarf den Inhalt der Nachricht und die Signatur und liefern einen Wahrheitswert bzw. den verschlüsselten Inhalt zurück.

(c) Übertragen Sie folgendes UML-Klassendiagramm in Programm-Code einer gängigen und geeigneten objektorientierten Sprache Ihrer Wahl. Die Methodenrümpfe dürfen Sie dabei leer lassen (auch wenn das Programm dann nicht übersetzbar bzw. ausführbar wäre).



Lösungsvorschlag

Interface „IMap“

```
interface IMap {
  Object get(String key);

  void put(String key, Object value);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2016/herbst/pki/IMap.java`

Klasse „MapImpl“

```
import java.util.ArrayList;
import java.util.List;

class MapImpl implements IMap {
  final static long serialVersionUID = 123;

  private List<MapEntry> entries;

  MapImpl() {
    entries = new ArrayList<MapEntry>();
  }

  public Object get(String key) {
    return new Object();
  }

  public void put(String key, Object value) {
    // Nicht verlangt in der Aufgabenstellung.
    entries.add(new MapEntry(key, value));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2016/herbst/pki/MapImpl.java`

Aktivitätsdiagramm
Klassendiagramm

Klasse „MapEntry“

```
class MapEntry {
  String key;
  Object value;

  MapEntry(String key, Object value) {
    // Nicht verlangt in der Aufgabenstellung.
    this.key = key;
    this.value = value;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2016/herbst/pki/MapEntry.java`

66116 / 2017 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 2

Gegeben sei das folgende Glossar, welches die statische Struktur von einfachen Aktivitätsdiagrammen in natürlicher Sprache beschreibt:

Aktivitätsdiagramm: Benannter Container für Aktivitäten und Datenflüsse. Eine der definierten Aktivitäten ist als Start-Aktivität ausgezeichnet.

Aktivität: Teil des beschriebenen Verhaltens. Man unterscheidet Start-, End-, echte Aktivitäten sowie Entscheidungen. Aktivitäten können generell mehrere ein- und auslaufende Kontrollflüsse haben.

Startaktivität: Ist im Aktivitätsdiagramm eindeutig und dient als Einstiegspunkt des beschriebenen Ablaufs.

Endaktivität: Wird eine solche Aktivität erreicht, ist der beschriebene Ablauf zu Ende.

Echte Aktivität: Benannte Aktion, die nach Ausführung zu einer definierten nächsten Aktivität führt.

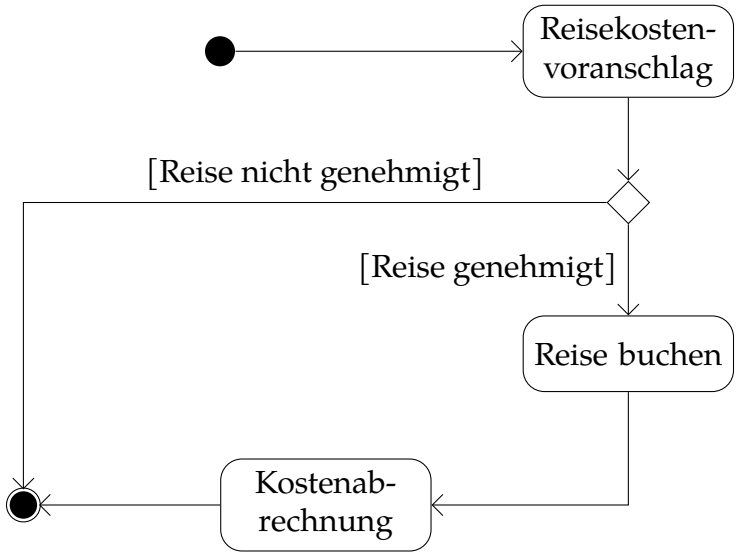
Entscheidung: Aktivität, die mehrere Nachfolger hat. Welche davon als nächstes ausgeführt wird, wird durch entsprechende Bedingungen (s. Kontrollfluss) gesteuert.

Kontrollfluss: Verbindet je eine Quell- mit einer Zielaktivität. Kann eine Bedingung enthalten, die erfüllt sein muss, damit die Zielaktivität im Falle einer Entscheidung ausgeföhrt wird.

(a) Geben Sie ein UML-Klassendiagramm an, welches die im Glossar definierten Konzepte und Beziehungen formal beschreibt. Geben Sie bei allen Attributen und Assoziationsenden deren Sichtbarkeit, Multiplizität imd Typ an. Benennen Sie alle Assoziationen.

Objektdiagramm

(b) Nachfolgend ist ein Beispiel eines Aktivitätsdiagramms in der gängigen grafischen Notation abgebildet. Stellen Sie den beschriebenen Kontrollfluss als UML-Objektdiagramm konform zum in Teilaufgabe a erstellten UML-Klassendiagramm dar. Referenzieren Sie die dort definierten Klassen und Assoziationen; auf Objektbezeichner dürfen Sie verzichten.



66116 / 2018 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 1

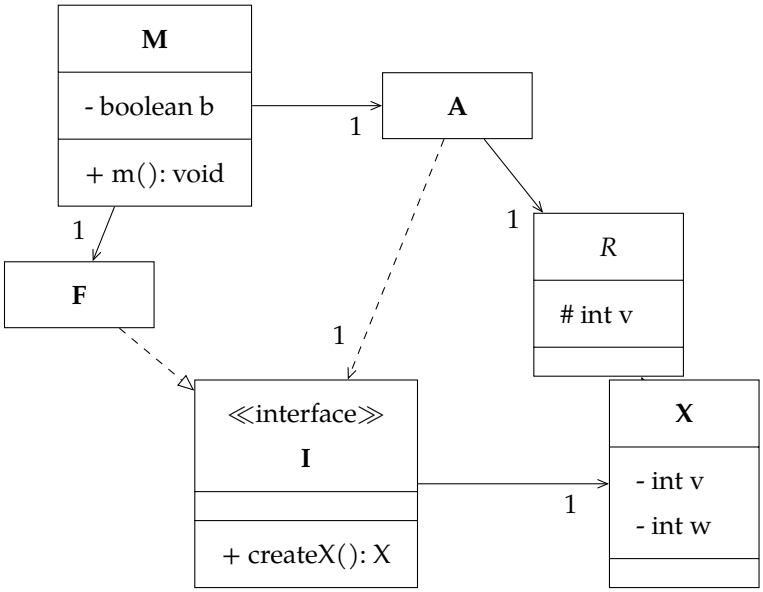
Gegeben sei das folgende Java-Programm:

```
class M {
  private boolean b;
  private F f;
  private A a;

  public void m() {
    f = new F();
    a = new A(f);
    b = true;
  }
}

class A {
  private R r;
  public A(I i) {
    r = i.createX();
  }
}

interface I {
  public X createX();
}

class F implements I {
  public X createX() {
    return new X(0, 0);
```

```
  }
}

abstract class R {
  protected int v;
}

class X extends R {
  private int v, w;
  public X(int v, int w) {
    this.v = v;
    this.w = w;
  }
}
```

(a) Das Subtypprinzip der objektorientierten Programmierung wird in obigem Programmcode zweimal ausgenutzt. Erläutern Sie wo und wie dies geschieht.

(b) Zeichnen Sie ein UML-Klassendiagramm, das die statische Struktur des obigen Programms modelliert. Instanzvariablen mit einem Klassentyp sollen durch gerichtete Assoziationen mit Rollennamen und Multiplizität am gerichteten Assoziationsende modelliert werden. Alle aus dem Programmcode ersichtlichen statischen Informationen (insbesondere Interfaces, abstrakte Klassen, Zugriffsrechte, benutzerdefinierte Konstruktoren und Methoden) sollen in dem Klassendiagramm abgebildet werden.

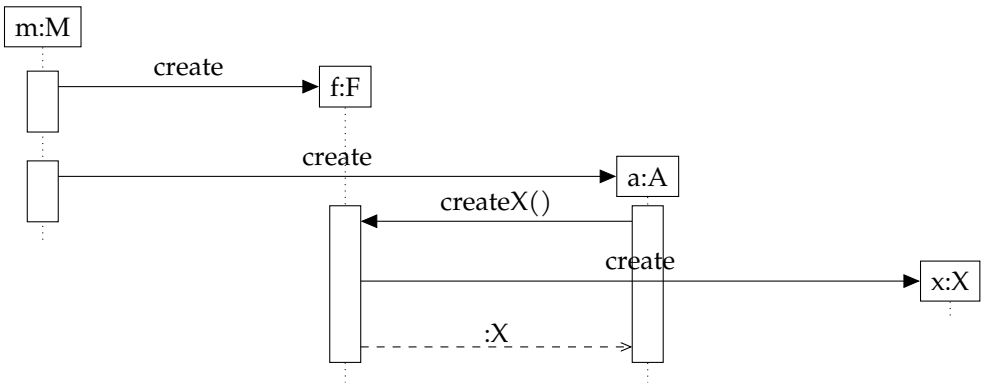


(c) Es wird angenommen, dass ein Objekt der Klasse M existiert, für das die Methode `m()` aufgerufen wird. Geben Sie ein Instanzendiagramm (Objektdiagramm) an, das alle nach der Ausführung der Methode `m` existierenden Objekte und deren Verbindungen (Links) zeigt.

(d) Wie in Teil c) werde angenommen, dass ein Objekt der Klasse M existiert, für das die Methode `m()` aufgerufen wird. Diese Situation wird in Abb. 1 dargestellt.

Sequenzdiagramm
Beobachter (Observer)

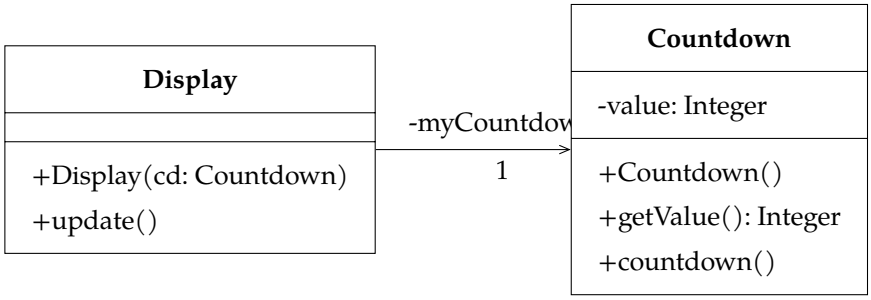
Zeichnen Sie ein Sequenzdiagramm, das Abb. 1 so ergänzt, dass alle auf den Aufruf der Methode `m()` folgenden Objekterzeugungen und Interaktionen gemäß der im Programmcode angegebenen Konstruktor- und Methodenrümpfe dargestellt werden. Aktivierungsphasen von Objekten sind durch längliche Rechtecke deutlich zu machen.



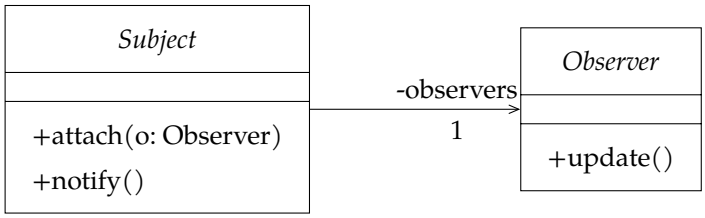
66116 / 2018 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 2

Es soll eine (kleine) Anwendung entwickelt werden, in der ein Zähler in 1-er Schritten von 5000 bis 0 herunterzählt. Der Zähler soll als Objekt der Klasse `Countdown` realisiert werden, die in UML-Notation dargestellt ist. Das Attribut value soll den aktuellen Zählerstand speichern, der mit dem Konstruktor zu initialisieren ist. Die Methode `getValue` soll den aktuellen Zählerstand liefern und die Methode `countdown` soll den Zähler von 5000 bis 0 herunterzählen.

Der jeweilige Zählerstand soll von einem Objekt der in untenstehender Abbildung angegebenen Klasse `Display` am Bildschirm ausgegeben werden. Bei der Konstruktion eines `Display`-Objekts soll es mit einem `Countdown`-Objekt verbunden werden, indem dessen Referenz unter `myCountdown` abgespeichert wird. Die Methode `update` soll den aktuellen Zählerstand vom `Countdown`-Objekt holen und mit `System.out.println` am Bildschirm ausgeben. Dies soll zu Beginn des Zählprozesses und nach jeder Änderung des Zählerstands erfolgen.



Damit das `Display`-Objekt über Zählerstände des `Countdown`-Objekts informiert wird, soll das Observer-Pattern angewendet werden. Untenstehende Abbildung zeigt die im Observer-Pattern vorkommenden abstrakten Klassen. (Kursivschreibweise bedeutet abstrakte Klasse bzw. abstrakte Methode.)



(a) Welche Wirkung haben die Methoden `attach` und `notify` gemäß der Idee des Observer-Patterns?

Lösungsvorschlag

Das beobachtete Objekt bietet mit der Methode `attach` einen Mechanismus, um Beobachter anzumelden und diese über Änderungen zu informieren.

Mit der Methode `notify` werden alle Beobachter benachrichtigt, wenn sich das beobachtete Objekt ändert.

(b) Welche der beiden Klassen `Display` und `Countdown` aus obenstehender Abbildung spielt die Rolle eines `Subject` und welche die Rolle eines `Observer`?

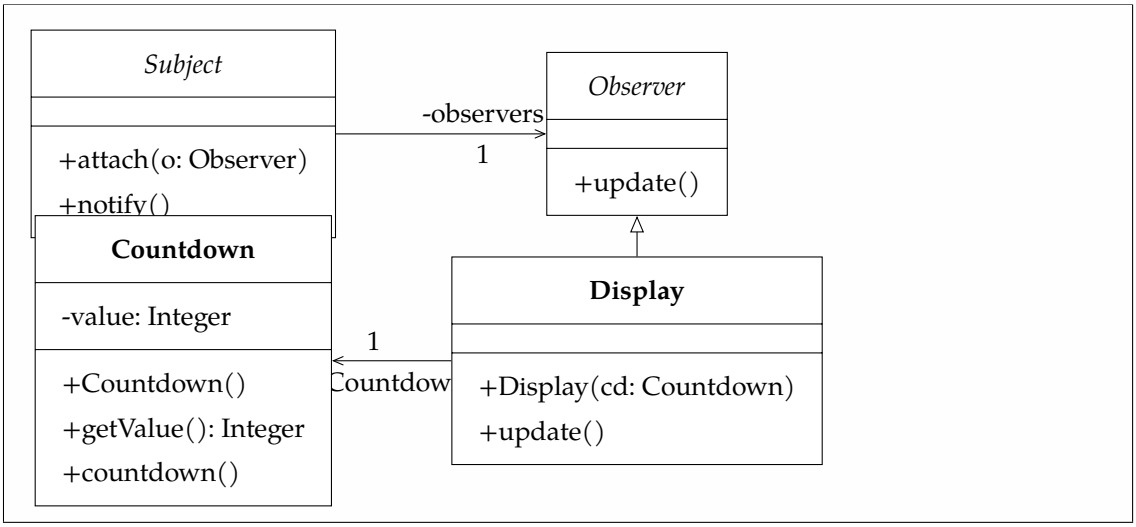
Lösungsvorschlag

Die Klasse `Countdown` spielt die Rolle des `Subject`s, also des Gegenstands, der beobachtet wird.

Die Klasse `Display` spielt die Rolle eines `Observer`, also die Rolle eines Beobachters.

(c) Erstellen Sie ein Klassendiagramm, das die beiden obenstehenden gegebenen Diagramme in geeigneter Weise, ðentsprechend der Idee des Observer-Patterns, zusammenfügt. Es reicht die Klassen und deren Beziehungen anzugeben. Eine nochmalige Nennung der Attribute und Methoden ist nicht notwendig.

Lösungsvorschlag



(d) Unsere Anwendung soll nun in einer objektorientierten Programmiersprache Ihrer Wahl (z. B. Java oder C++) implementiert werden. Dabei soll von folgenden Annahmen ausgegangen werden:

- Das Programm wird mit einer main-Methode gestartet, die folgenden Rumpf hat:

```
public static void main(String[] args){
  Countdown cd = new Countdown();
  new Display(cd);
  cd.countdown();
}
```

- Die beiden Klassen Subject und Observer sind bereits gemäß der Idee des Observer-Patterns implementiert.

Geben Sie auf dieser Grundlage eine Implementierung der beiden Klassen `Display` und `Countdown` an, so dass das gewünschte Verhalten, ðAnzeige der Zählerstände und Herunterzählen des Zählers, realisiert wird. Die Methoden der Klassen `Subject` und `Observer` sind dabei auf geeignete Weise zu verwenden bzw. zu implementieren. Geben Sie die verwendete Programmiersprache an.

Lösungsvorschlag

```
public class Client {
  public static void main(String[] args){
    Countdown cd = new Countdown();
    new Display(cd);
    cd.countdown();
    cd.countdown();
    cd.countdown();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Client.java`

```
import java.util.ArrayList;
import java.util.List;

public abstract class Subject {
  private final List<Observer> observers = new ArrayList<Observer>();

  public void attach(Observer o) {
    observers.add(o);
  }

  public void notifyObservers() {
    for (Observer o : observers) {
      o.update();
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Subject.java`

```
public abstract class Observer {
  public abstract void update();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Observer.java`

```
public class Countdown extends Subject {

  private int value;

  public Countdown() {
    value = 5000;
  }

  public int getValue() {
    return value;
  }

  public void countdown() {
    if (value > 0) {
      notifyObservers();
      value--;
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Countdown.java`

```
public class Display extends Observer {
  Countdown myCountdown;
  public Display(Countdown cd) {
    myCountdown = cd;
    myCountdown.attach(this);
  }

  public void update() {
    System.out.println(myCountdown.getValue());
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/fruehjahr/Display.java`

66116 / 2018 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 2

Gegeben sei das Java-Programm:

```
public class Music {
  private Beatle state;

  public Music() {
    state = new Paul();
  }

  public void help() {
    this.state.help(this);
  }

  public void obladi() {
```

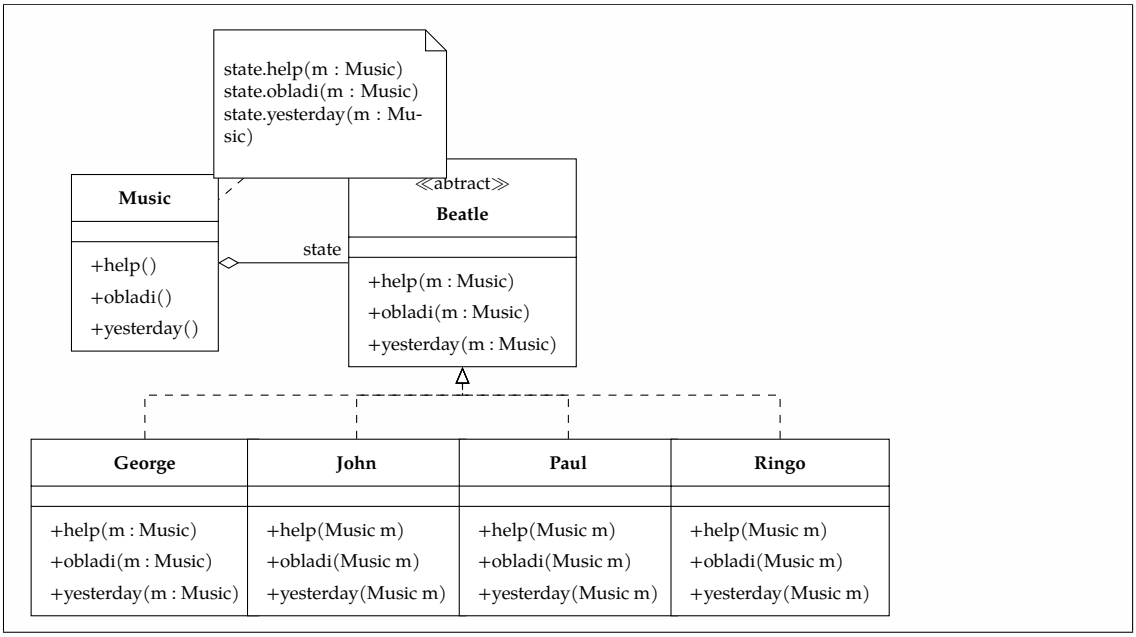
```
    this.state.obladi(this);
  }

  public void yesterday() {
    this.state.yesterday(this);
  }

  public void setBeatle(Beatle b) {
    state = b;
  }
}

abstract class Beatle {
  public abstract void help(Music m);

  public abstract void obladi(Music m);

  public abstract void yesterday(Music m);
}

class George extends Beatle {
  public void help(Music m) {
    System.out.println("help");
    m.setBeatle(new John());
  }

  public void obladi(Music m) {
  }

  public void yesterday(Music m) {
    System.out.println("yesterday");
    m.setBeatle(new Paul());
  }
}

class John extends Beatle {
  public void help(Music m) {
    System.out.println("help");
    m.setBeatle(new Paul());
  }

  public void obladi(Music m) {
    System.out.println("obladi");
    m.setBeatle(new Ringo());
  }

  public void yesterday(Music m) {
  }
}

class Paul extends Beatle {
  public void help(Music m) {
    System.out.println("help");
    m.setBeatle(new George());
  }
```

```
  public void obladi(Music m) {
  }

  public void yesterday(Music m) {
    System.out.println("yesterday");
  }
}

class Ringo extends Beatle {
  public void help(Music m) {
    System.out.println("help");
    m.setBeatle(new John());
  }

  public void obladi(Music m) {
  }

  public void yesterday(Music m) {
    System.out.println("yesterday");
    m.setBeatle(new Paul());
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2018/herbst/beatles/Music.java`

(a) Zeichnen Sie ein UML-Klassendiagramm, das die statische Struktur des Programms modelliert. Instanzvariablen mit einem Klassentyp sollen durch gerichtete Assoziationen mit Rollennamen und passender Multiplizität am gerichteten Assoziationsende modelliert werden. Alle aus dem Programmcode ersichtlichen statischen Informationen sollen in dem Klassendiagramm dargestellt werden.

Lösungsvorschlag

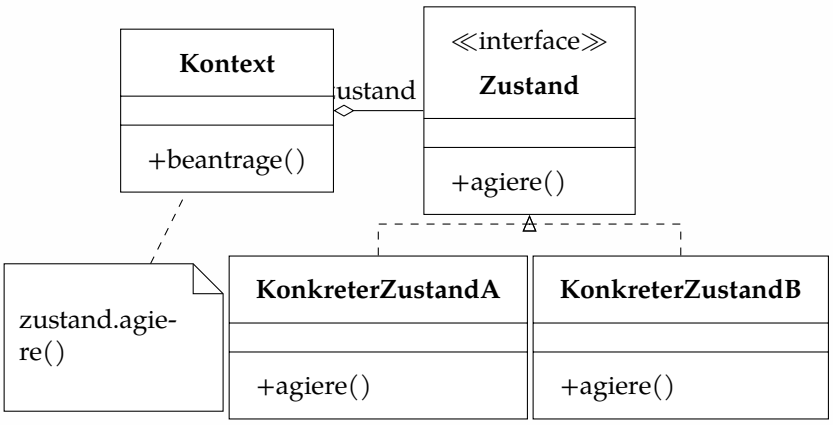


Das Programm implementiert ein Zustandsdiagramm, das das Verhalten von Objekten der Klasse `Music` beschreibt. Für die Implementierung wurde das Design-Pattern `STATE` angewendet.

Zustand (State)
Zustandsdiagramm zeichnen

(c) Geben Sie die statische Struktur des STATE-Patterns an und erläutern Sie, welche Rollen aus dem Entwurfsmuster den Klassen des gegebenen Programms dabei zufallen und welche Operationen aus dem Entwurfsmuster durch (ggf. mehrere) Methoden in unserem Beispielprogramm implementiert werden. Es ist von den z. B. im Design-Pattern-Katalog von Gamma et al. verwendeten Namen auszugehen, das heißt von Klassen mit Namen `Context`, `State`, `ConcreteStateA`, `ConcreteStateB` und von Operationen mit Namen `request` und `handle`.

Lösungsvorschlag



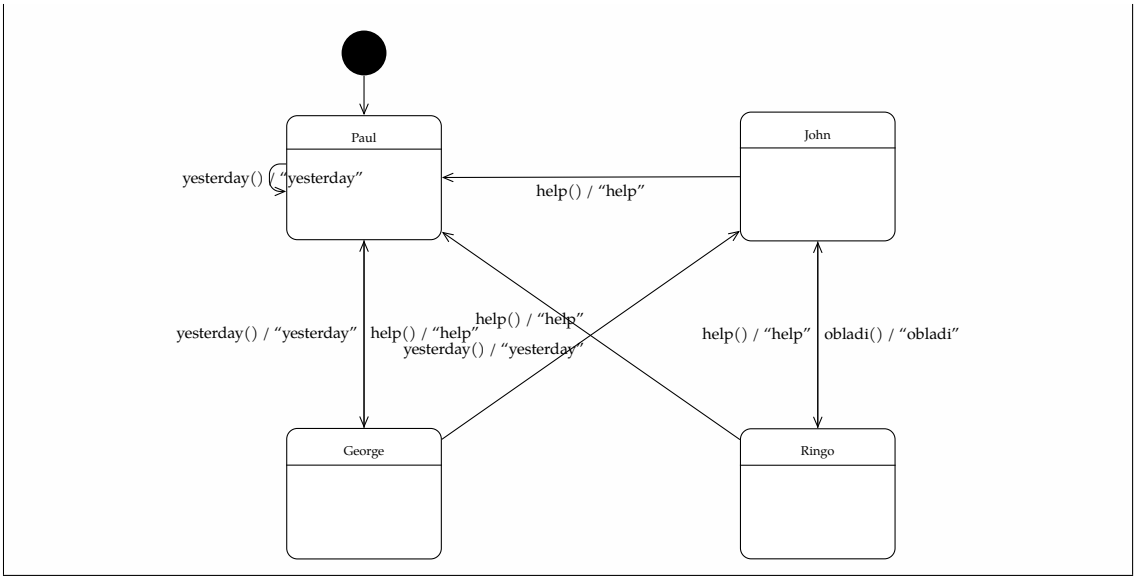
Kontext (Context) definiert die clientseitige Schnittstelle und verwaltet die separaten Zustandsklassen.

State (Zustand) definiert eine einheitliche Schnittstelle aller Zustandsobjekte und implementiert gegebenenfalls ein Standardverhalten.

KontreterZustand (ConcreteState) implementiert das Verhalten, das mit dem Zustand des Kontextobjektes verbunden ist.

(d) Zeichnen Sie das UML-Zustandsdiagramm (mit Anfangszustand), das von dem Programm implementiert wird. Dabei muss - gemäß der UML-Notation - unterscheidbar sein, was Ereignisse und was Aktionen sind. In dem Diagramm kann zur Vereinfachung statt `System.out.println ("x")` einfach `"x"` geschrieben werden.

Lösungsvorschlag



66116 / 2019 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 1

Gegeben sei folgender Sachverhalt: Eine Grafik ist entweder ein Kreis, ein Quadrat oder ein Dreieck. Eine Grafik kann zudem auch eine Kombination aus diesen Elementen sein. Des Weiteren können Sie aus mehreren Grafiken auch neue Grafiken zusammenbauen. Sie denken sich: Ich möchte eine Menge von Grafiken genauso wie eine einzelne Grafik behandeln können.

(a) Welches Entwurfsmuster sollten Sie zur Modellierung verwenden?

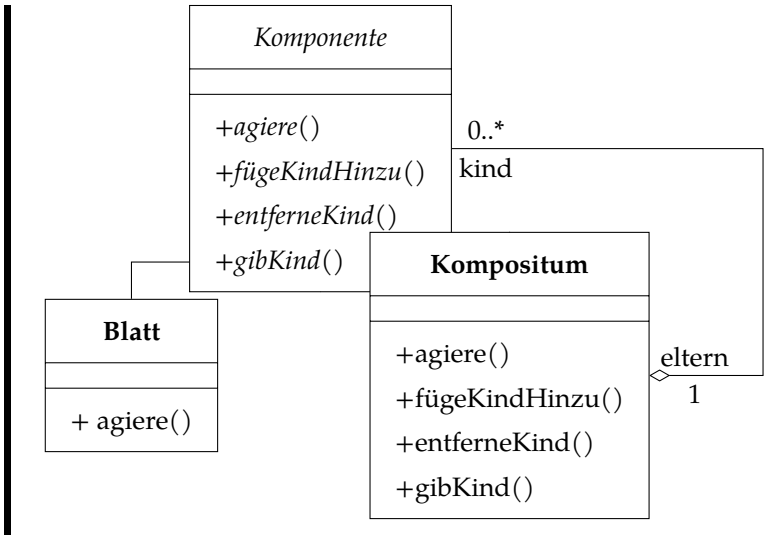
Lösungsvorschlag

Kompositum

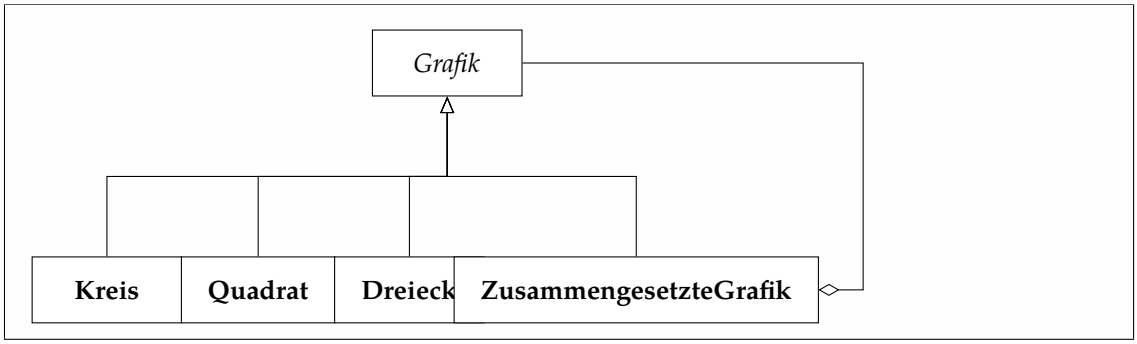
(b) Zeichnen Sie das entsprechende Klassendiagramm. Es reicht, nur die Klassennamen mit ihren Assoziationen und Vererbungsbeziehungen anzugeben; ðohne Methoden und Attribute.

Exkurs: Kompositum

Klassendiagramm



Lösungsvorschlag



66116 / 2019 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 2

Hintergrundinformationen für die Aufgaben 2-5: Modellierung und Implementierung eines Programms

Ein autonomer Roboter in einer Montagehalle bekommt einen Auftrag, eine Menge von Objekten aus dem Lager (Material, z. B. Schrauben oder Werkzeug, z. B. Bohrer) zu holen und anschließend an seinen Ausgangspunkt zu bringen. Der Roboter hat einen Namen, der ihn eindeutig identifiziert, kennt seine aktuelle Position (x- und y-Koordinate) und hat als weitere Eigenschaft den Auftrag, der aus einer Liste von Auftragspositionen besteht, die er holen soll. Der Roboter besitzt folgende Fähigkeiten (Methoden):

(a) Er kann sich zu einer angegebenen Position bewegen.

(b) Er hat eine Methode, um einen Auftrag abzuarbeiten, d.h. alle Auftragspositionen zu holen.

Alle Objekte im Lager haben jeweils einen Namen, einen Standort mit Angabe einer Position (s. oben) und speichern außerdem die jeweils noch vorhandene Stückzahl. Es gibt zwei Typen von Objekten: Werkzeuge mit einer Methode, mit der ein einzelnes Werkzeug ausgeliehen werden kann, und Materialien mit einer Methode, mit der sie verbraucht werden, wobei eine gewünschte Stückzahl angegeben wird. Diese vorgegebenen Methoden aktualisieren die Stückzahlen der Werkzeuge (Reduktion um 1) bzw. Materialien (Reduktion um verbrauchte Stückzahl), wobei hier vereinfachend angenommen wird, dass die Stückzahlen der Werkzeuge und Materialien immer ausreichend

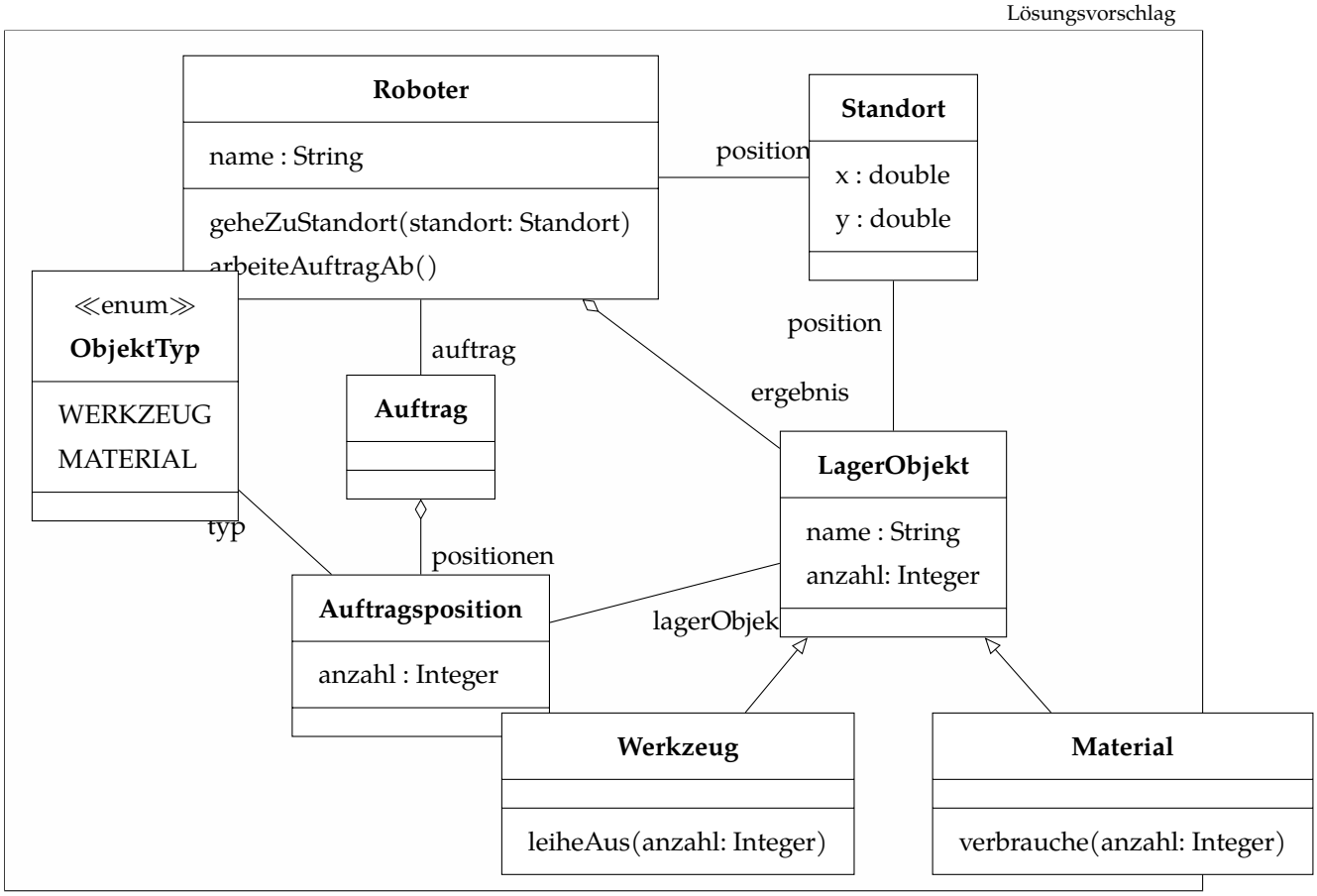
groß sind, um die geforderten Mengen bedienen zu können (d.h. Sie können diese beiden Methoden nutzen, ohne deren Implementierung angeben zu müssen).

Ein Auftrag (z. B. *„Hole Bohrer Typ B1, 100 × Schrauben M6, 10 × Schrauben M10 und 2 × Blech B72“*) besteht aus einer Menge von Auftragspositionen. Eine Auftragsposition besteht aus dem Typ des zu holenden Objekts (Werkzeug oder Material, soll als Enumeration modelliert werden), einem Verweis auf das zu holende Objekt und der zu holenden Stückzahl (Quantität; bei Werkzeugen wird diese ignoriert, da sie immer 1 ist). Der Roboter soll über die Auftragsposition außerdem die Position bestimmen, zu der er fahren muss, um das Objekt zu holen.

Der Roboter arbeitet die Auftragspositionen in der Reihenfolge ab, indem er sich von seinem aktuellen Standpunkt immer zur am nächsten liegenden Auftragsposition bewegt, um dort das nächste Objekt zu holen. Wir gehen der Einfachheit halber davon aus, dass die Montagehalle gut aufgeräumt ist und der Roboter sich quasi entlang der Luftlinie bewegen kann, d.h. die Entfernung zwischen zwei Positionen entspricht der euklidischen Distanz (Wurzel aus der Summe der Quadrate der Differenzen der x- und y-Koordinaten der Positionen). Der Roboter soll zur Kontrolle als weitere Eigenschaft „Ergebnis“ die Liste der eingesammelten Objekte zu einem Auftrag in der Reihenfolge speichern, in der er sie geholt hat, z.B. (M6 M10 B72 B1).

Geben Sie ein UML-Klassendiagramm zu der Aufgabenstellung an. Hinweis: Bei den Aufgaben 4 und 5 wird Konsistenz des Aktivitätsdiagramms bzw. des Codes mit dem Klassendiagramm verlangt.

Lösungsvorschlag



66116 / 2019 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 4

(a) Geben Sie für alle Methoden, die zur Abarbeitung des Auftrages für den Roboter erforderlich sind, Pseudocode an. Nutzen Sie (im Klassendiagramm definierte)

Aktivitätsdiagramm
Klassendiagramm
Objektdiagramm

Hilfsmethoden, um den Code übersichtlicher zu gestalten. (Verwenden Sie statt einer komplizierten Methode mehrere einfachere Methoden, die sich aufrufen.)

Achten Sie auf die Konsistenz zum Klassendiagramm (Inkonsistenzen führen zu Punktabzügen).

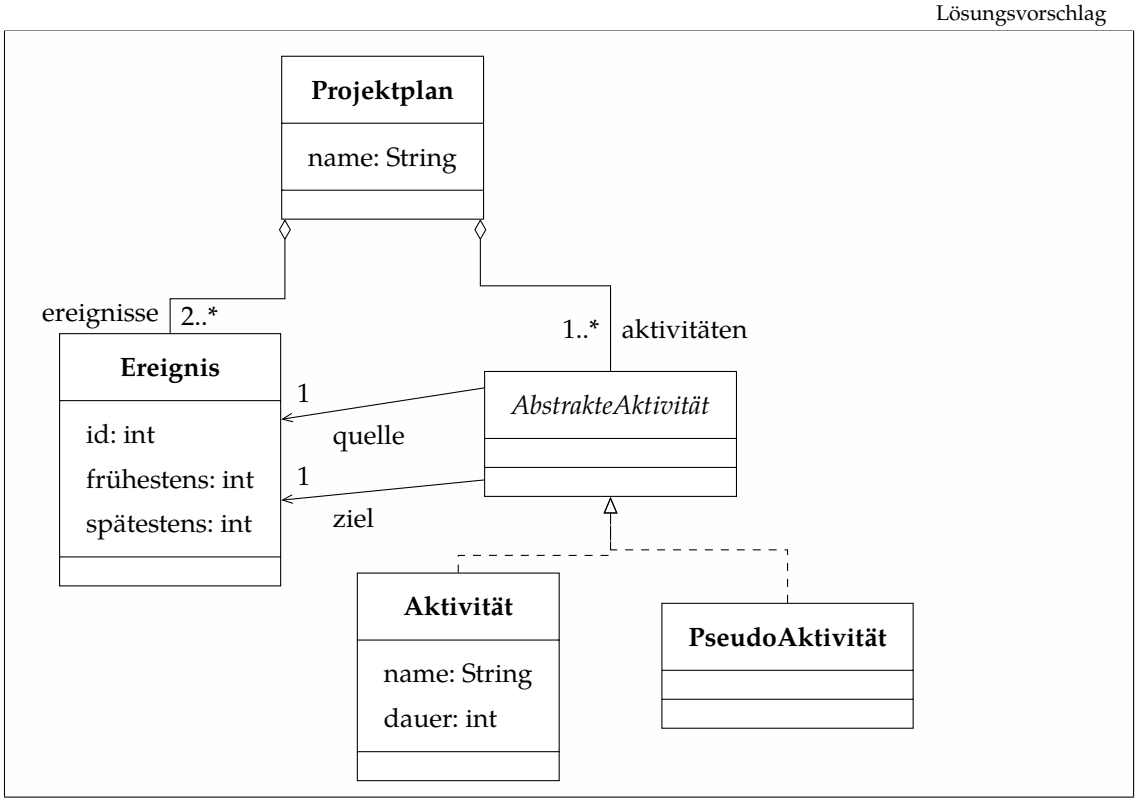
(b) Überführen Sie den Pseudocode der Hauptmethode zur Abarbeitung eines Auftrags in ein syntaktisch korrektes UML-Aktivitätsdiagramm. Schleifen und Verzweigungen müssen durch strukturierte Knoten dargestellt werden (d. h. kein graphisches "goto").

66116 / 2019 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 1

Ein CPM-Netzwerk („Critical Path Method") ist ein benannter Projektplan, der aus Ereignissen und Aktivitäten besteht. Ein Ereignis wird durch eine ganze Zahl > 0 identifiziert. Jede Aktivität führt von einem Quellereignis zu einem Zielereignis. Eine reale Aktivität hat einen Namen und eine Dauer (eine ganze Zahl > 0). Eine Pseudoaktivität ist anonym. Ereignisse und Pseudoaktivitäten verbrauchen keine Zeit. Zu jedem Ereignis gibt es einen frühesten und einen spätesten Zeitpunkt (eine ganze Zahl > 0), deren Berechnung nicht Gegenstand der Aufgabe ist.

(a) Erstellen Sie ein UML-Klassendiagramm zur Modellierung von CPM-Netzwerken. Geben Sie für Attribute jeweils den Namen und den Typ an. Geben Sie für Assoziationen den Namen und für jedes Ende den Rollennamen und die Multiplizität an. Nutzen Sie ggf. abstrakte Klassen, Vererbung, Komposition oder Aggregation. Verzichten Sie auf Operationen und Sichtbarkeiten.

Lösungsvorschlag



(b) Erstellen Sie für das Klassendiagramm aus a) und das Beispiel aus der Aufgabenstellung ein Objektdiagramm. Geben Sie Rollennamen nur an, wenn es notwendig ist, um die Enden eines Links (Instanz einer Assoziation) zu unterscheiden.

UML-Diagramme
Anwendungsfalldiagramm
Aktivitätsdiagramm
Zustandsdiagramm Wissen
Zustand (State)

66116 / 2019 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 2

Eine Dienstreise an einer Universität wird folgendermaßen abgewickelt:

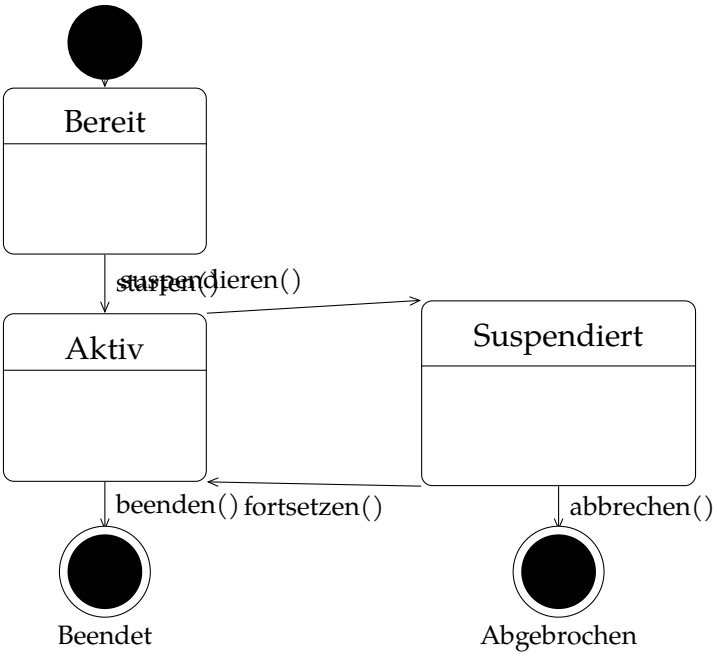
Ein Mitarbeiter erstellt einen Dienstreiseantrag und legt ihn seinem Vorgesetzten zur Unterschrift vor. Mit seiner Unterschrift befürwortet der Vorgesetzte die Dienstreise. Verweigert er die Unterschrift, wird der Vorgang abgebrochen. Nach der Befürwortung wird der Antrag an die Reisekostenstelle weitergeleitet, die über die Annahme des Antrags entscheidet. Im Falle einer Ablehnung wird der Vorgang abgebrochen; sonst genehmigt die Reisekostenstelle den Antrag. Der Mitarbeiter kann nun einen Abschlag beantragen. Stimmt der Vorgesetzte zu, so entscheidet die Reisekostenstelle über den Abschlag und weist ggf. die Kasse der Universität an, den Abschlag auszuzahlen. Nach Ende der Dienstreise erstellt der Mitarbeiter einen Antrag auf Erstattung der Reisekosten an die Reisekostenstelle. Die Reisekostenstelle setzt den Erstattungsbetrag fest und weist die Kasse an, den Betrag auszuzahlen.

(a) Erstellen Sie ein Anwendungsfalldiagramm (Use-Case-Diagramm) für Dienstreisen.

(b) Erstellen Sie ein Aktivitätsdiagramm, das den oben beschriebenen Ablauf modelliert. Ordnen Sie den Aktionen die Akteure gemäß a) zu. Beschränken Sie sich auf den Kontrollfluss (keine Objektflüsse).

66116 / 2019 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 4

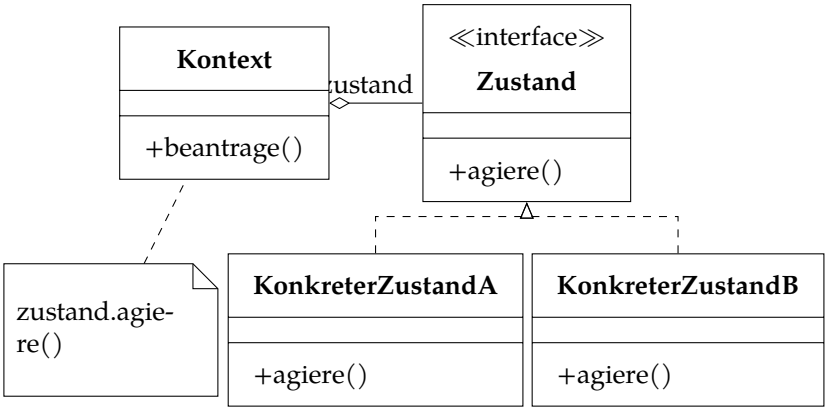
Zu den Aufgaben eines Betriebssystems zählt die Verwaltung von Prozessen. Jeder Prozess durchläuft verschiedene Zustände; Transitionen werden durch Operationsaufrufe ausgelöst. Folgendes Zustandsdiagramm beschreibt die Verwaltung von Prozessen:



Implementieren Sie dieses Zustandsdiagramm in einer Programmiersprache Ihrer Wahl mit Hilfe des Zustandsmusters; geben Sie die gewählte Sprache an. Die Methoden für die Transitionen sollen dabei die Funktionalität der Prozessverwaltung simulieren, indem der Methodenaufruf auf der Standardausgabe protokolliert wird. Falls Transitionen im aktuellen Zustand undefiniert sind, soll eine Fehlermeldung ausgegeben werden.

Exkurs: Zustand-(State)-Entwurfsmuster

UML-Klassendiagramm



Teilnehmer

Kontext (Context) definiert die clientseitige Schnittstelle und verwaltet die separaten Zustandsklassen.

State (Zustand) definiert eine einheitliche Schnittstelle aller Zustandsobjekte und implementiert gegebenenfalls ein Standardverhalten.

KontreterZustand (**ConcreteState**) implementiert das Verhalten, das mit dem Zustand des Kontextobjektes verbunden ist.

Lösungsvorschlag

Implementierung in der Programmiersprache „Java“:

Methoden	Zustände	Klassennamen
	Bereit	ZustandBereit
starten(), fortsetzen()	Aktiv	ZustandAktiv
suspendieren()	Suspendiert	ZustandSuspendiert
beenden()	Beendet	ZustandBeendet
abbrechen()	Abgebrochen	ZustandAbgebrochen

```
/**
 * Entspricht der „Kontext"-Klasse in der Terminologie der „Gang of
 * Four".
 */
public class Prozess {

  private ProzessZustand aktuellerZustand;

  public Prozess() {
    aktuellerZustand = new ZustandBereit(this);
  }

  public void setzeZustand(ProzessZustand zustand) {
    aktuellerZustand = zustand;
  }

  public void starten() {
    aktuellerZustand.starten();
  }

  public void suspendieren() {
    aktuellerZustand.suspendieren();
  }

  public void fortsetzen() {
    aktuellerZustand.fortsetzen();
  }

  public void beenden() {
    aktuellerZustand.beenden();
  }

  public void abbrechen() {
    aktuellerZustand.abbrechen();
  }
```

```
  public static void main(String[] args) {
    Prozess prozess = new Prozess();
    prozess.starten();
    prozess.suspendieren();
    prozess.fortsetzen();
    prozess.beenden();
    prozess.starten();

    // Ausgabe:
    // Der Prozess ist im Zustand „bereit"
    // Der Prozess wird gestartet.
    // Der Prozess ist im Zustand „aktiv"
    // Der Prozess wird suspendiert.
    // Der Prozess ist im Zustand „suspendiert"
    // Der Prozess wird fortgesetzt.
    // Der Prozess ist im Zustand „aktiv"
    // Der Prozess wird beendet.
    // Der Prozess ist im Zustand „beendet"
    // Im Zustand „beendet" kann die Transition „starten" nicht ausführt werden!
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/Prozess.java`

```
/**
 * Entspricht der „Zustand"-Klasse in der Terminologie der "Gang of
 * Four".
 */
abstract class ProzessZustand {

  Prozess prozess;

  String zustand;

  public ProzessZustand(String zustand, Prozess prozess) {
    this.zustand = zustand;
    this.prozess = prozess;
    System.out.println(String.format("Der Prozess ist im Zustand „%s"", zustand));
  }

  private void gibFehlermeldungAus(String transition) {
    System.err.println(

↪   String.format("Im Zustand „%s" kann die Transition „%s" nicht ausführt werden!",
            zustand, transition));
  }

  public void starten() {
    gibFehlermeldungAus("starten");
  }

  public void suspendieren() {
    gibFehlermeldungAus("suspendieren");
  }
```

```
  public void fortsetzen() {
    gibFehlermeldungAus("fortsetzen");
  }

  public void beenden() {
    gibFehlermeldungAus("beenden");
  }

  public void abbrechen() {
    gibFehlermeldungAus("abbrechen");
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ProzessZustand.java`

```
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der "Gang of
 * Four".
 */
public class ZustandAbgebrochen extends ProzessZustand {

  public ZustandAbgebrochen(Prozess prozess) {
    super("abgebrochen", prozess);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandAbgebrochen.java`

```
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
 * Four".
 */
public class ZustandAktiv extends ProzessZustand {

  public ZustandAktiv(Prozess prozess) {
    super("aktiv", prozess);
  }

  public void suspendieren() {
    System.out.println("Der Prozess wird suspendiert.");
    prozess.setzeZustand(new ZustandSuspendiert(prozess));
  }

  public void beenden() {
    System.out.println("Der Prozess wird beendet.");
    prozess.setzeZustand(new ZustandBeendet(prozess));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandAktiv.java`

```
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
```

```
 * Four".
 */
public class ZustandBeendet extends ProzessZustand {

  public ZustandBeendet(Prozess prozess) {
    super("beendet", prozess);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandBeendet.java`

```
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
 * Four".
 */
public class ZustandBereit extends ProzessZustand {

  public ZustandBereit(Prozess prozess) {
    super("bereit", prozess);
  }

  public void starten() {
    System.out.println("Der Prozess wird gestartet.");
    prozess.setzeZustand(new ZustandAktiv(prozess));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandBereit.java`

```
/**
 * Entspricht der „KonkreterZustand"-Unterklasse in der Terminologie der „Gang of
 * Four".
 */
public class ZustandSuspendiert extends ProzessZustand {

  public ZustandSuspendiert(Prozess prozess) {
    super("suspendiert", prozess);
  }

  public void fortsetzen() {
    System.out.println("Der Prozess wird fortgesetzt.");
    prozess.setzeZustand(new ZustandAktiv(prozess));
  }

  public void abbrechen() {
    System.out.println("Der Prozess wird abgebrochen.");
    prozess.setzeZustand(new ZustandAbgebrochen(prozess));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/prozess_verwaltung/ZustandSuspendiert.java`

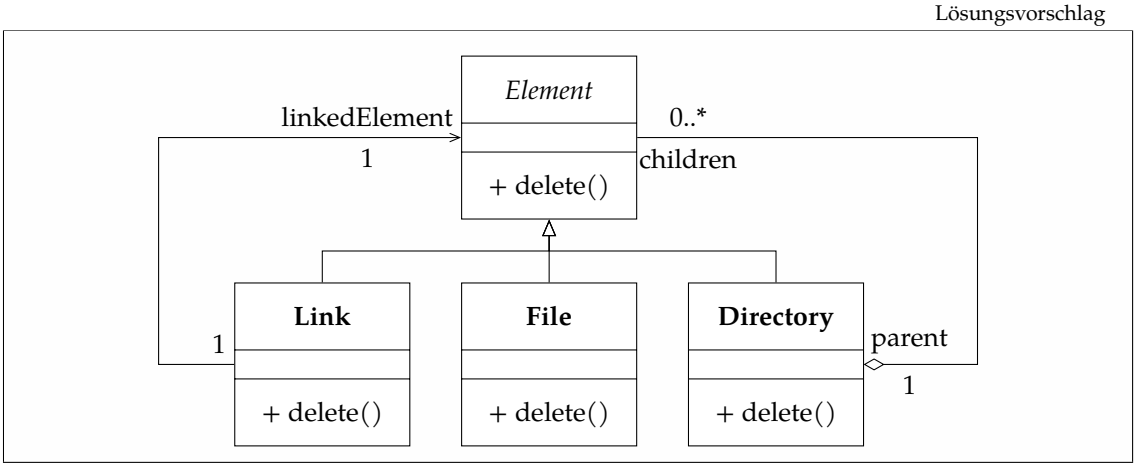
66116 / 2019 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 1

Entwurfsmuster
Kompositum (Composite)
Klassendiagramm
Implementierung in Java

Hierarchische Dateisysteme bestehen aus den FileSystemElements Ordner, Dateien und Verweise. Ein Ordner kann seinerseits Ordner, Dateien und Verweise beinhalten; jedem Ordner ist bekannt, welche Elemente (children) er enthält. Mit Ausnahme des Root-Ordners auf der obersten Hierarchieebene ist jeder Ordner, jede Datei und jeder Verweis Element eines Elternordners. Jedem Element ist bekannt, was sein Elternordner ist (parent). Ein Verweis verweist auf einen Verweis, eine Datei oder einen Ordner (link). Wenn ein Ordner gelöscht wird, werden alle seine Bestandteile ggf. rekursiv ebenfalls gelöscht. Sie dürfen die Lösungen für Aufgabenteil b) und c) in einem gemeinsamen Code kombinieren.

(a) Modellieren Sie diesen Sachverhalt mit einem UML-Klassendiagramm. Benennen Sie die Rollen von Assoziationen und geben Sie alle Kardinalitäten an. Ihre Lösung soll mindestens eine sinnvolle Spezialisierungsbeziehung enthalten.

Lösungsvorschlag



(b) Implementieren Sie das Klassendiagramm als Java- oder C++-Programm. Jedes Element des Dateisystems soll mindestens über ein Attribut `name` verfügen. Übergeben Sie den Elternordner jedes Elements als Parameter an den Konstruktor; der Elternordner des Root-Ordners kann dabei als `null` implementiert werden. Dokumentieren Sie Ihren Code.

Lösungsvorschlag

a

```
public abstract class Element {

  protected String name;

  protected Element parent;

  protected Element(String name, Element parent) {
    this.name = name;
    this.parent = parent;
  }

  public String getName() {
    return name;
```

```
  }

  public abstract void delete();

  public abstract boolean isDirectory();

  public abstract void addChild(Element child);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Element.java`

```
import java.util.ArrayList;
import java.util.List;

public class Directory extends Element {
  private List<Element> children;

  public Directory(String name, Element parent) {
    super(name, parent);
    children = new ArrayList<Element>();
    if (parent != null)
      parent.addChild(this);
  }

  public void delete() {
    System.out.println("The directory "" + name +
↪   "" was deleted and it's children were also deleted.");
    for (int i = 0; i < children.size(); i++) {
      Element child = children.get(i);
      child.delete();
    }
  }

  public void addChild(Element child) {
    children.add(child);
  }

  public boolean isDirectory() {
    return true;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Directory.java`

```
public class File extends Element {

  public File(String name, Element parent) {
    super(name, parent);
    parent.addChild(this);
  }

  public void delete() {
    System.out.println("The File "" + name + "" was deleted.");
  }
```

```
  public boolean isDirectory() {
    return false;
  }

  /**
   * Eine Datei kann keine Kinder haben. Deshalb eine Methode mit leerem
   * Methodenrumpf.
   */
  public void addChild(Element child) {
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/File.java`

```
public class Link extends Element {

  private Element linkedElement;

  public Link(String name, Element parent, Element linkedElement) {
    super(name, parent);
    this.linkedElement = linkedElement;
    parent.addChild(this);
  }

  public void delete() {
    System.out.println("The Symbolic Link "" + name + "" was deleted.");
    linkedElement.delete();
    System.out.println("The linked element "" + name + "" was deleted too.");
  }

  public void addChild(Element child) {
    if (linkedElement.isDirectory())
      linkedElement.addChild(child);
  }

  public boolean isDirectory() {
    return linkedElement.isDirectory();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2019/herbst/file_system/Link.java`

[a] TU Darmstadt: Dr. Michael Eichberg - Case Study Using the Composite and Proxy Design Patterns

(c) Ordnen Sie eine Methode `delete`, die Dateien, Ordner und Verweise rekursiv löscht, einer oder mehreren geeigneten Klassen zu und implementieren Sie sie. Zeigen Sie die Löschung jedes Elements durch eine Textausgabe von `name` an. `toString()` müssen Sie dabei nicht implementieren. Gehen Sie davon aus, dass Verweis- und Ordnerstrukturen azyklisch sind und dass jedes Element des Dateisystems höchstens einmal existiert. Wenn ein Verweis gelöscht wird, wird sowohl der Verweis als auch das verwiesene Element bzw. transitiv die Kette der

verwiesenen Elemente gelöscht. Bedenken Sie, dass die Löschung eines Elements immer auch Konsequenzen für den dieses Element beinhaltenden Ordner hat. Es gibt keinen Punktabzug, wenn Sie die Löschung des Root-Ordners nicht zulassen. Rekursion

Lösungsvorschlag

Siehe Antwort zu Aufgabe b)

(d) Was kann im Fall von `delete` passieren, wenn die Linkstruktur zyklisch ist oder die Ordnerstruktur zyklisch ist? Kann es zu diesen Problemen auch dann führen, wenn weder die Linkstruktur zyklisch ist, noch die Ordnerstruktur zyklisch ist? Wie kann man im Programm das Problem lösen, falls man Zyklizitäten zulassen möchte?

Lösungsvorschlag

Falls die Link- oder Ordnerstruktur zyklisch ist, kann es aufgrund der Rekursion zu einer Endlosschleife kommen. Diese Problem tritt bei azyklischen Strukturen nicht auf, weil der rekursive Löschvorgang beim letzten Element abgebrochen wird (Abbruchbedingung).

Das Problem kann zum Beispiel durch ein neues Attribut `gelöscht` in der Klasse `Link` oder `Directory` gelöst werden. Dieses Attribut wird auf `true` gesetzt, bevor es in die Rekursion einsteigt. Rufen sich die Klassen wegen der Zyklizität selbst wieder auf, kommt es durch entsprechende IF-Bedingungen zum Abbruch.

Außerdem ist ein Zähler denkbar, der sich bei jeder Rekursion hochsetzt und ab einem gewissen Grenzwert zum Abbruch führt.

(e) Was ist ein Design Pattern? Nennen Sie drei Beispiele und erläutern Sie sie kurz. Welches Design Pattern bietet sich für die Behandlung von hierarchischen Teil-Ganzes-Beziehungen an, wie sie im Beispiel des Dateisystems vorliegen?

Lösungsvorschlag

Design Pattern sind wiederkehrende, geprüfte, bewährte Lösungsschablonen für typische Probleme.

Drei Beispiele

Einzelstück (Singleton) Stellt sicher, dass nur *genau eine Instanz einer Klasse* erzeugt wird.

Beobachter (Observer) Das Observer-Muster ermöglicht einem oder mehreren Objekten, automatisch auf die *Zustandsänderung* eines bestimmten Objekts *zu reagieren*, um den eigenen Zustand anzupassen.

Stellvertreter (Proxy) Ein Proxy stellt einen Platzhalter für eine andere Komponente (Objekt) dar und kontrolliert den Zugang zum echten Objekt.

Für hierarchischen Teil-Ganzes-Beziehungen eigent sich das Kompositum (Com-

posite). Es ermöglicht die Gleichbehandlung von Einzelelementen und Elementgruppierungen in einer verschachtelten Struktur (z. B. Baum), sodass aus Sicht des Clients keine explizite Unterscheidung notwendig ist.

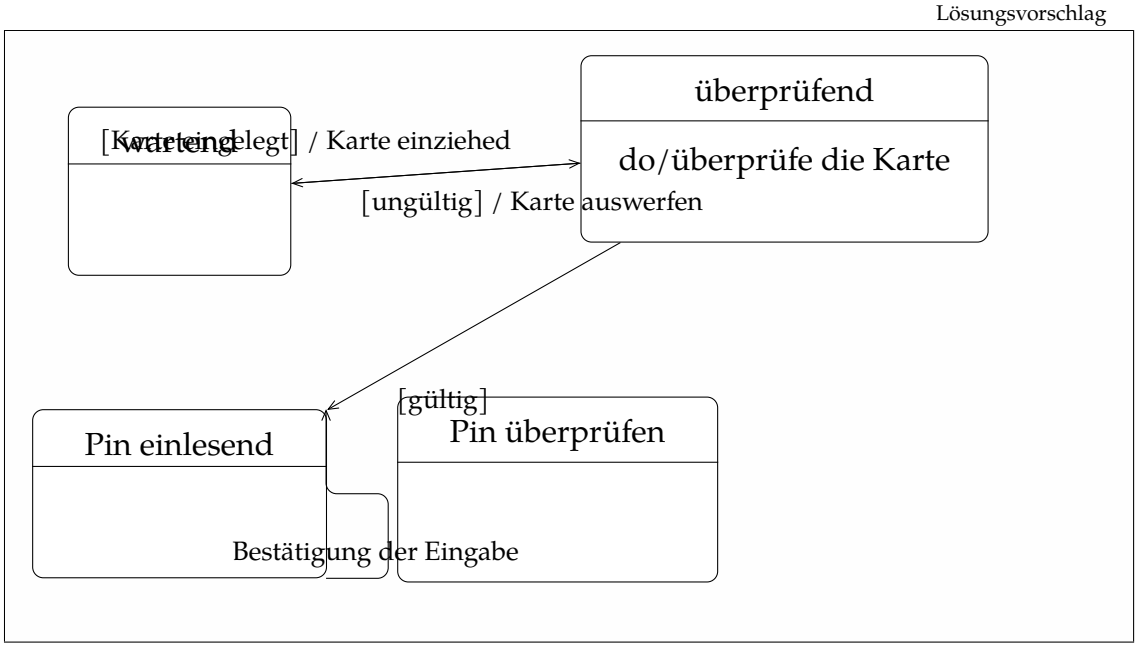
66116 / 2020 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 1

(a) Basisfunktion

Erstellen Sie ein Zustandsdiagramm für einen Bankautomat, welcher den im Folgenden beschriebenen Authentifizierungsvorgang von Bankkunden realisiert. Modellieren Sie dazu soweit nötig sowohl Zustände und Transitionsbedingungen als auch die Aktionen der Zustände.

Der Bankautomat startet im Grundzustand und wartet auf das Einlegen einer Bankkarte. Wird eine Karte eingelegt, wird diese eingezogen und der Automat startet die Überprüfung der Bankkarte. Ist die Karte ungültig, wird die Karte ausgeworfen und der Automat wechselt in den Grundzustand. Ist die Karte gültig, kann die vierstellige PIN eingelesen werden. Nach der Bestätigung der Eingabe wird diese überprüft. Ist die PIN gültig, so stoppt der Automat und zeigt eine erfolgreiche Authentifizierung an. Ist die PIN ungültig, zeigt der Automat einen Fehler an und erlaubt eine erneute Eingabe der PIN.

Lösungsvorschlag



(b) Erweiterung

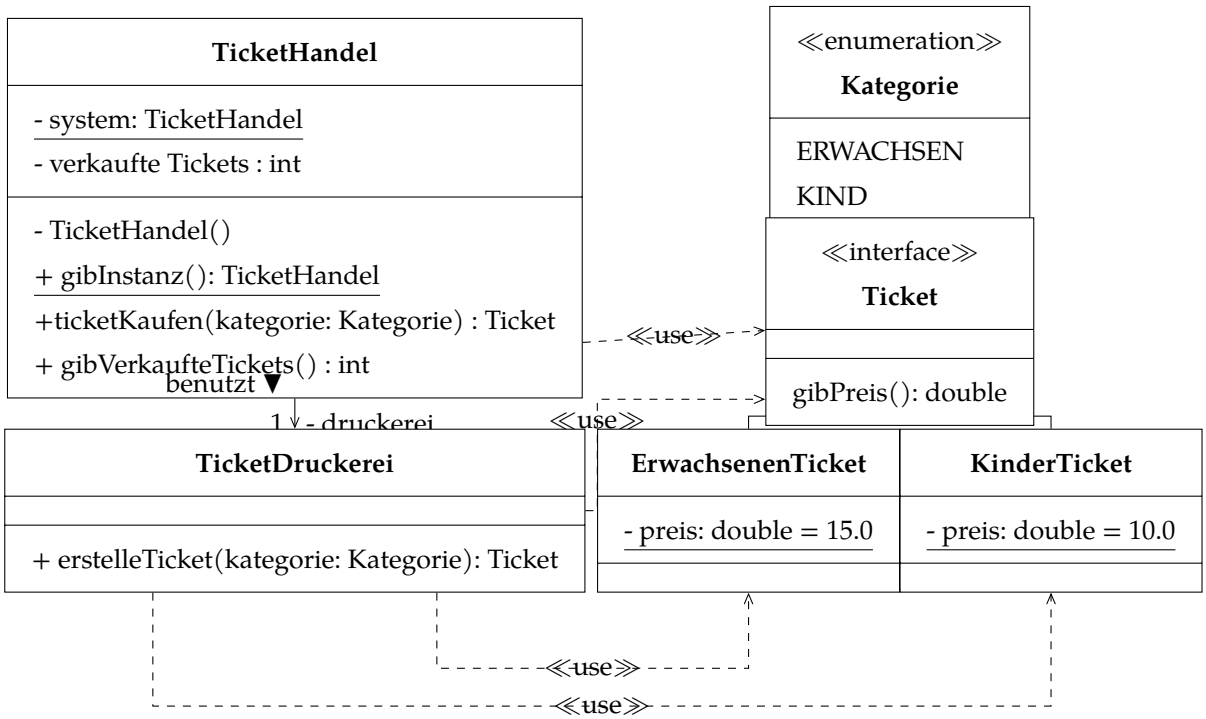
Der Bankautomaten aus Aufgabe a) soll nun so verändert werden, dass ein Bankkunde nach dem ersten fehlerhaften Eingeben der PIN die PIN erneut eingeben muss. Bei erneuter Falscheingabe, wird eine dritte Eingabe möglich. Bei der dritten Falscheingabe der PIN wird die Karte vom Automaten eingezogen und der Automat geht wieder in den Ausgangszustand über.

Hinweis: Für diese Aufgabe dürfen Sie Ihr Zustandsdiagramm aus a) weiter verwenden, wenn Sie eindeutig, z. B. durch den Einsatz von Farben kennzeichnen,

was nur zur Aufgabe a) gehört und was Abänderungen des Zustandsdiagramms aus a) sind. Sie können, falls Sie einen neuen Automaten zeichnen, Zustände und Übergänge, die inhaltsgleich zur Lösung des Aufgabenteils a) sind mit einem "W" markieren, statt sie zu beschriften. In diesem Fall wird der Text aus der Lösung zu Aufgabenteil a) an dieser Stelle wiederholt gedacht.

Entwurfsmuster
Einzelstück (Singleton)
Fabrikmethode (Factory Method)

66116 / 2020 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 3

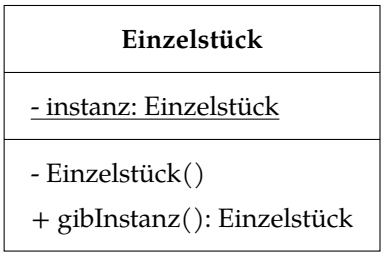


Ihnen sei ein UML-Klassendiagramm zu folgendem Szenario gegeben. Ein Benutzer (nicht im Diagramm enthalten) kann über einen `TicketHandel` Tickets erwerben. Dabei muss der Benutzer eine der zwei Ticketkategorien angeben. Das Handelsystem benutzt eine `TicketDruckerei`, um ein passendes `Ticket` für den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

Lösungsvorschlag

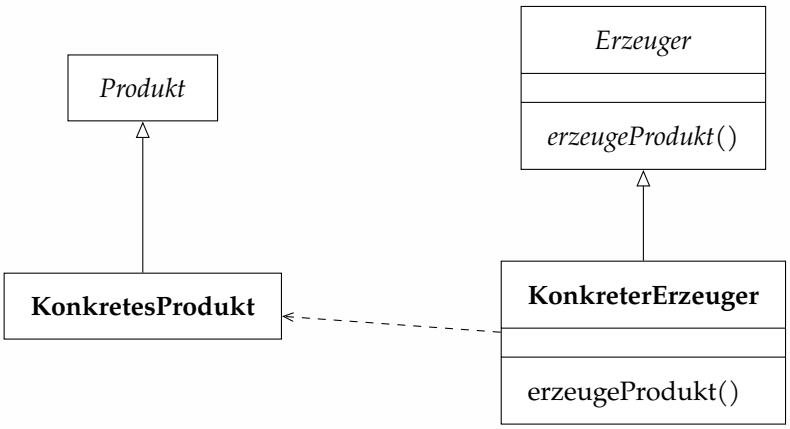
Einzelstück (Singleton)



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.

Lösungsvorschlag

- 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.

Lösungsvorschlag

```
public class TicketHandel {
  private static TicketHandel system;
  private int verkaufteTickets;
  private TicketDruckerei druckerei;

  private TicketHandel() {
    druckerei = new TicketDruckerei();
    verkaufteTickets = 0;
  }

  public static TicketHandel gibInstanz() {
    if (system == null) {
      system = new TicketHandel();
    }
    return system;
  }

  public Ticket ticketKaufen(Kategorie kategorie) {
    verkaufteTickets++;
    return druckerei.erstelleTicket(kategorie);
  }

  public int gibVerkaufteTickets() {
    return verkaufteTickets;
  }
}
```

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.

Lösungsvorschlag

```java
public class TicketDruckerei {
  public Ticket erstelleTicket(Kategorie kategorie) {
    if (kategorie == Kategorie.ERWACHSEN) {
      return new ErwachsenenTicket();
    } else {
      return new KinderTicket();
    }
  }
}
```

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.

Lösungsvorschlag

```java
public class KinderTicket implements Ticket {
  private static double preis = 10.0;

  public double gibPreis() {
    return preis;
  }
}
```

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:

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

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

Konsolenausgabe:

Anzahl verkaufter Tickets: 2

66116 / 2020 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 4

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.

Lösungsvorschlag

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.

Lösungsvorschlag

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.

Lösungsvorschlag

- 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.

Lösungsvorschlag

- Nachricht
- Preis

- Rabatt

Sequenzdiagramm
Entwurfsmuster

(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.

66116 / 2020 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 5

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?

Lösungsvorschlag

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`

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

  public double auswerten() {
    return Klient.symbole.get(name);
  }

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

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

```
abstract class Term {
  abstract public double auswerten();

  abstract public void ausgeben();
}
```

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

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 3

Wählen Sie bis zu fünf unterschiedliche Elementtypen aus folgendem Diagramm aus und benennen Sie diese Elemente und ihre syntaktische (nicht semantische) Bedeutung.

Lösungsvorschlag

Klasse ConcreteCreator

Interface Produkt

Abstrakte Klasse Creator

Kommentar return new ConcreteProdukt()

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 6

(a) Erläutern Sie den Zweck (Intent) des Erzeugungsmusters Erbauer in max. drei Sätzen, ohne dabei auf die technische Umsetzung einzugehen.

Lösungsvorschlag

> Die Erzeugung komplexer Objekte wird vereinfacht, indem der Konstruktionsprozess in eine spezielle Klasse verlagert wird. Er wird so von der Repräsentation getrennt und kann sehr unterschiedliche Repräsentationen zurückliefern.

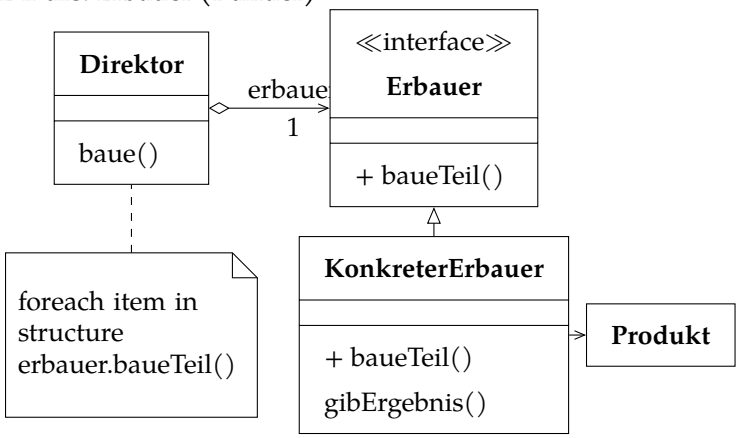
(b) Erklären Sie, wie das Erzeugungsmuster Erbauer umgesetzt werden kann (Implementierung). Die Angabe von Code ist hierbei NICHT notwendig!

(c) Nennen Sie jeweils einen Vor- und einen Nachteil des Erzeugungsmusters Erbauer im Vergleich zu einer Implementierung ohne dieses Muster.

Lösungsvorschlag

> **Vorteil** Die Implementierungen der Konstruktion und der Repräsentationen werden isoliert. Die Erbauer verstecken ihre interne Repräsentation vor dem Direktor.
>
> **Nachteil** Es besteht eine enge Kopplung zwischen Produkt, konkretem Erbauer und den am Konstruktionsprozess beteiligten Klassen.

Exkurs: Erbauer (Builder)



Erbauer Der Erbauer spezifiziert eine abstrakte Schnittstelle zur Erzeugung der Teile eines komplexen Objektes.

KonkreterErbauer Der konkrete Erbauer erzeugt die Teile des komplexen Objekts durch Implementierung der Schnittstelle. Außerdem definiert und verwaltet er die von ihm erzeugte Repräsentation des Produkts. Er bietet auch eine Schnittstelle zum Auslesen des Produkts.

Direktor Der Direktor konstruiert ein komplexes Objekt unter Verwendung der Schnittstelle des Erbauers. Der Direktor arbeitet eng mit dem Erbauer zusammen: Er weiß, welche Baureihenfolge der Erbauer verträgt oder benötigt. Der Direktor entkoppelt somit den Konstruktionsablauf vom Klienten.

Produkt Das Produkt repräsentiert das zu konstruierende komplexe Objekt.

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 7

Lesen Sie die folgenden alternativen Codestücke.

(a)
```
public class MyParser {
  private InputStream input;

  public MyParser(String filePath) {

  }
}
```

(b)
```
public class MyParser {
  private InputStream input;

  public MyParser(InputStream stream) {
  }
}
```

Beide Codestücke zeigen die Initialisierung einer Klasse namens `MyParser`. Das zweite Codestück nutzt jedoch hierfür eine Technik namens Abhängigkeits-Injektion (Dependency Injection).

(a) Erklären Sie den Unterschied zwischen beiden Initialisierungen. Hinweis: Sie können diese Aufgabe auch lösen, falls Sie die Technik nicht kennen.

Lösungsvorschlag

> Die Abhängigkeit von einer Instanz der Klasse InputStream wird erst bei der Initialisierung des Objekt übergeben.

(b) Benennen Sie einen Vorteil dieser Technik.

Lösungsvorschlag

> Die Kopplung zwischen einer Klasse und ihrer Abhängigkeit wird verringert.

66116 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 4

(a) Nennen und erklären Sie kurz die drei Kategorien der Organisation von klassischen Entwurfsmustern. Geben Sie zu jeder Kategorie ein Beispiel-Pattern an.

(b) Erstellen Sie ein Klassendiagramm zu den Komponenten einer Beobachtungsstation, welche aus einem einzigen Wasserstandsmesser besteht. Dabei sollen bei Änderungen des Wasserstandes der aktuelle Wasserstand sowohl auf der Konsole als auch in eine Log-Datei geschrieben werden. Der momentane Wasserstand soll dabei mittels einer Variablen des Datentyps double dargestellt werden. Die verschiedenen Anzeigearten (Konsolenanzeige, Logger) sollen als verschiedene

Klas- sen modelliert werden und enthalten jeweils nur eine Methode zum Anzeigen bzw. Schreiben des aktuellen Wasserstandes. Verwenden Sie für die Realisierung dieses Klassendiagramms die pas- senden Entwurfsmuster.

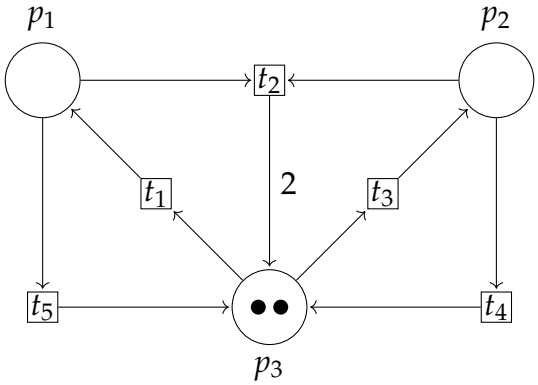
Hinweise:

- Getter und Setter müssen nicht eingezeichnet werden.
- Fügen Sie auch Methoden ein, welche durch die einzelnen Klassen implementiert werden.

Projektplanung

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4

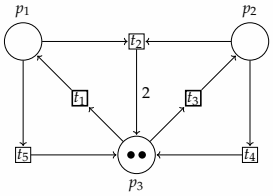
Gegeben Sei das folgende Petri-Netz:



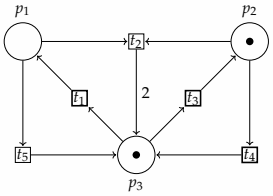
(a) Zeichnen Sie den Erreichbarkeitsgraphen des Petri-Netzes.

Lösungsvorschlag

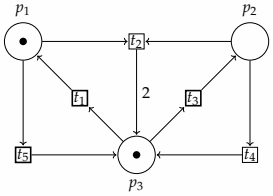
(0,0,2):



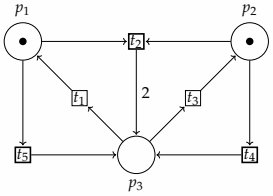
(0,1,1):

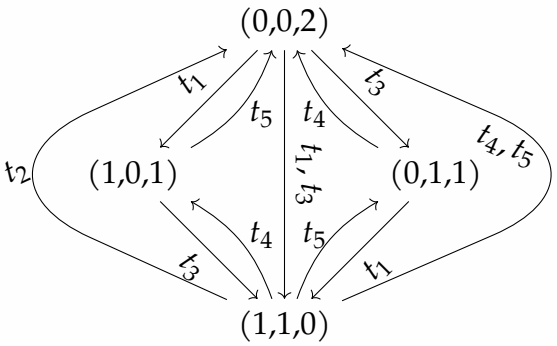


(1,0,1):



(1,1,0):





(b) Begründen Sie anhand des Erreichbarkeitsgraphen, ob das Petri-Netz lebendig, umkehrbar und/oder verklemmungsfrei ist. Petri-Netz

Lösungsvorschlag

lebenig Ja. Es gibt im Erreichbarkeitsgraphen keine Senke, also keinen Zustand, aus dem man nicht mehr heraus kommt.

umkehrbar Im Erreichbarkeitsgraphen kommt man von (0,0,2) auf verschiedenen Wegen wieder zurück zu (0,0,2). Man kann sich unendlich oft im Graph bewegen. Die Anfangsmarkierung kann wieder erreicht werden ($t_1 \to t_3 \to t_2$ oder $t_1 \to t_3 \to t_5 \to t_4$).

verklemmungsfrei Ja. Es gibt im Erreichbarkeitsgraphen keine Senke, also keinen Zustand, aus dem man nicht mehr herauskommt.

(c) Geben Sie die Darstellungsmatrix A sowie den Belegungsvektor v an und berechnen Sie damit die Belegung nach Schaltung von $t_1 \to t_3 \to t_2$.

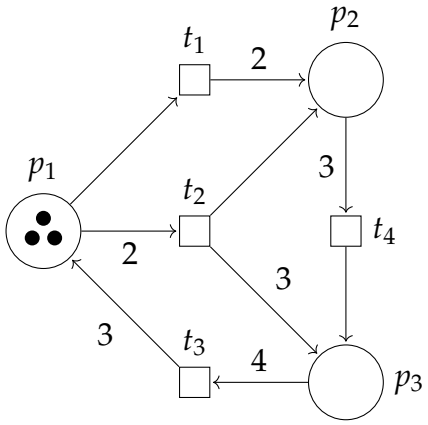
Lösungsvorschlag

$$A = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{ccccc} t_1 & t_2 & t_3 & t_4 & t_5 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 & -1 \\ 0 & -1 & 1 & -1 & 0 \\ -1 & 2 & -1 & 1 & 1 \end{pmatrix} \end{array}, \quad v = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}$$

$$v_{\text{neu}} = v + A \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + A \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + A \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix} = v$$

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4

Gegeben ist das folgende Petri-Netz:



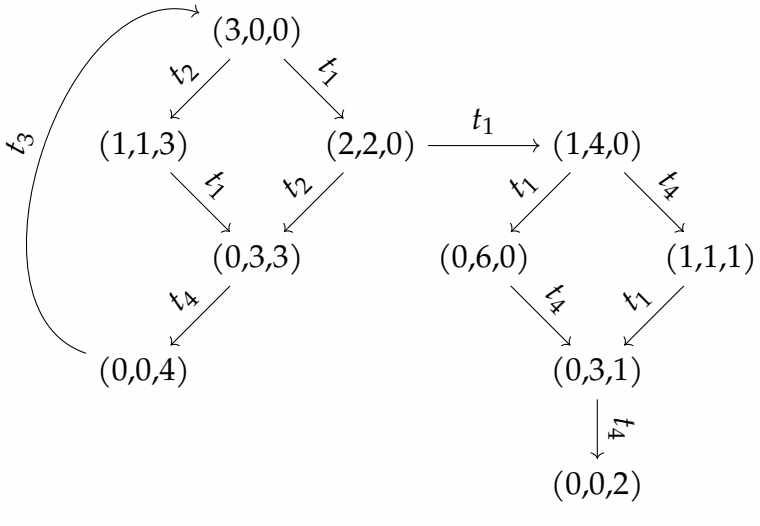
(a) Geben Sie die dazugehörige Darstellungsmatrix sowie den Belegungsvektor an.

Lösungsvorschlag

$$A = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} -1 & -2 & 3 & 0 \\ 2 & 1 & 0 & -3 \\ 0 & 3 & -4 & 1 \end{pmatrix} \end{array}, \quad v = \begin{pmatrix} 3 \\ 0 \\ 0 \end{pmatrix}$$

(b) Skizzieren Sie den Erreichbarkeitsgraphen des Petri-Netzes.

Lösungsvorschlag



(c) Begründen Sie anhand des Erreichbarkeitsgraphen, ob das Petri-Netz verklemmungsfrei ist oder nicht.

Lösungsvorschlag Petri-Netz
Petri-Netz

Durch Schalten von $t_1 \rightarrow t_1 \rightarrow t_1 \rightarrow t_4 \rightarrow t_4$ wird beispielsweise eine Verklemmung erreicht. Das Petri-Netz ist also nicht verklemmungsfrei. Am Erreichbarkeitsgraphen erkennt man das anhand der Senke im Knoten $[0,0,2]$.

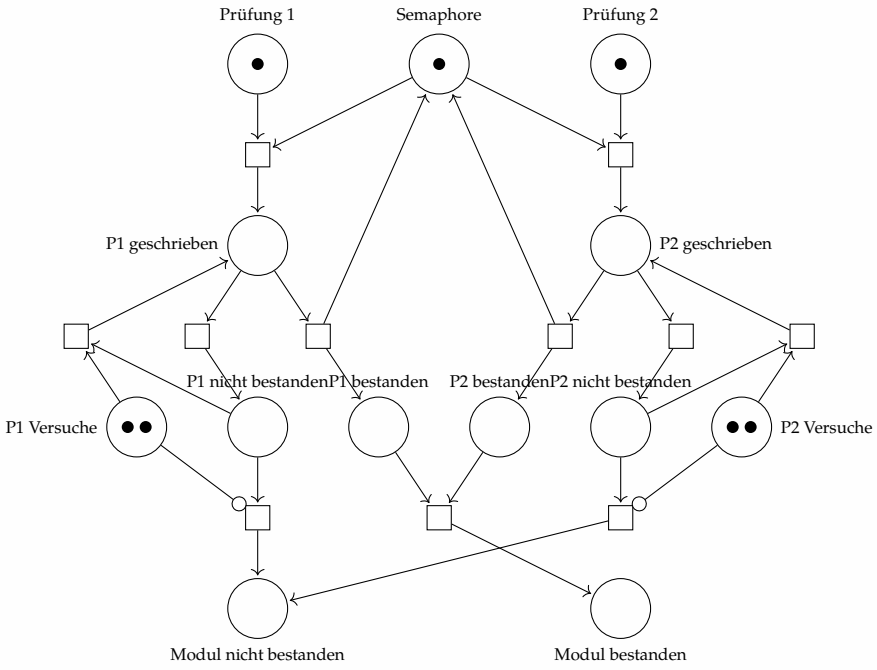
66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4

Modellieren Sie folgendes Szenario als Petri-Netz:

Ein Modul gilt als bestanden, wenn beide Prüfungen P_1 und P_2 bestanden sind. Beide Prüfungen dürfen bei Nicht-Bestehen jeweils maximal zwei mal wiederholt werden. Die Prüfungen dürfen nicht gleichzeitig geschrieben werden. Erst wenn eine von beiden bestanden wurde, darf die nächste begonnen werden. Wurde eine der beiden Prüfungen insgesamt drei mal nicht bestanden, so gilt das gesamte Modul als nicht bestanden.

Lösungsvorschlag

Man beachte die sog. Inhibitorkanten Linien mit Punkt am Ende). Z. B. kann t 7 nur schalten, wenn die Stelle „Wdh. Versuche P1“ keine Markierung enthält.



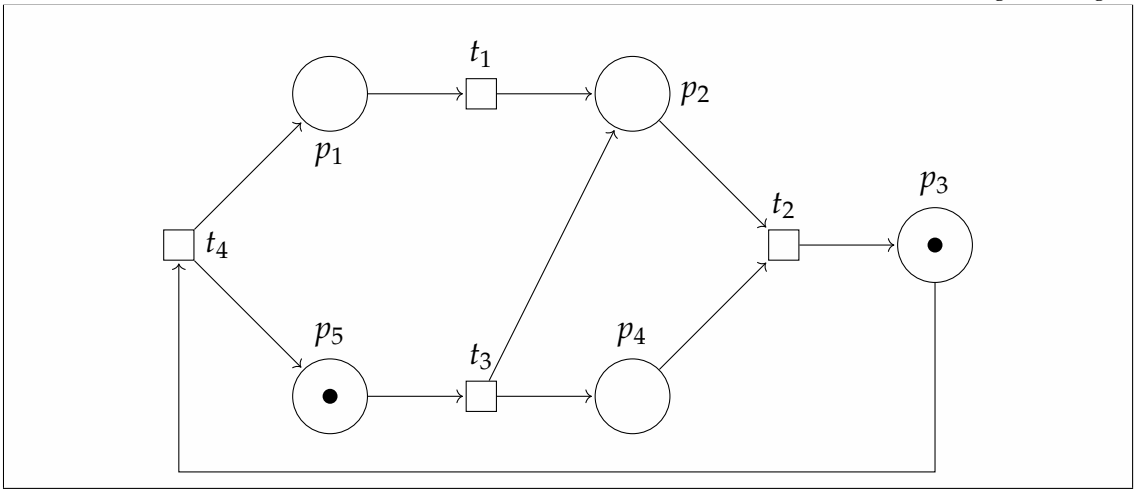
66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4

Gegeben sei die Darstellungsmatrix A und der Belegungsvektor v eines Petri-Netzes:

$$A = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \end{array}, \quad v = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

(a) Skizzieren Sie das zugehörige Petri-Netz.

Lösungsvorschlag



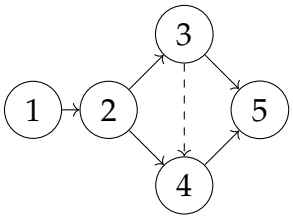
(b) Berechnen Sie mithilfe der Darstellungsmatrix A und zum Belegungsvektor v, die Belegung nach Schaltung von $t_3 \to t_2 \to t_4$.

Lösungsvorschlag

$$v_{\text{neu}} = v + A \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + A \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + A \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4

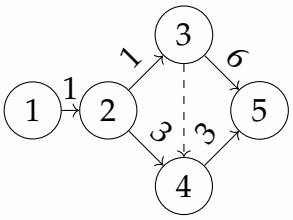
(a) Gegeben ist folgender (unvollständiger) CPM-Netzplan, sowie die frühesten und spätesten Termine und die Pufferzeiten aller Ereignisse:



Ereignis	1	2	3	4	5
frühester Termin	0	1	2	4	8
spätester Termin	0	1	2	5	8
Puffer	0	0	0	1	0

Vervollständigen Sie den CPM-Netzplan, indem Sie mit Hilfe obiger Tabelle die Zeiten der Vorgänge berechnen.

Lösungsvorschlag



Frühester Termin/Zeitpunkt

— Wir führen eine Vorwärtsterminierung durch und addieren die Dauern. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Maximum aus. **Erläuterungen:** i: Ereignis i; FZ_i: Frühester Zeitpunkt, zu dem Ereignis i eintreten kann. ———

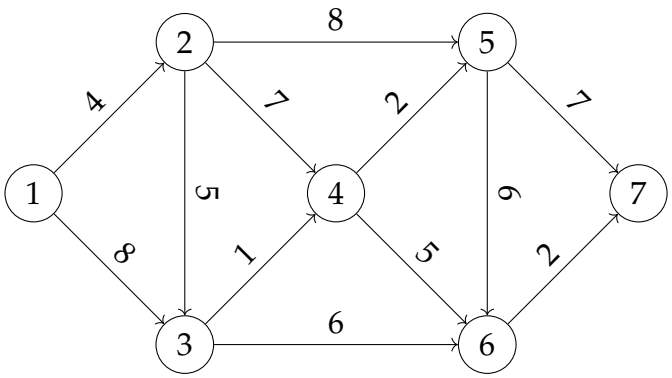
i	Nebenrechnung	FZ_i
1		0
2		1
3		2
4	$\max(4_2, 2_3)$	4
5	$\max(8_3, 7_4)$	8

Spätester Termin/Zeitpunkt

— Wir führen eine Rückwärtsterminierung durch und subtrahieren die Dauern vom letzten Ereignis aus. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Minimum aus. **Erläuterungen:** i: Ereignis i; SZ_i: Spätester Zeitpunkt, zu dem Ereignis i eintreten kann. ———

i	Nebenrechnung	SZ_i
5	siehe FZ_5	8
4		5
3	$\min(2_5, 5_4)$	2
2	$\min(1_3, 2_4)$	1
1		0

(b) Bestimmen Sie zum nachfolgenden CPM-Netzplan für jedes Ereignis den *frühesten Termin*, den *spätesten Termin* sowie die *Gesamtpufferzeit*. Geben Sie außerdem den *kritischen Pfad* an.



Lösungsvorschlag

i	1	2	3	4	5	6	7
FZ_i	0	4	9	11	13	19	21
SZ_i	0	4	10	11	13	19	21
GP	0	0	1	0	0	0	0

Frühester Termin/Zeitpunkt

i	Nebenrechnung	FZ_i
1		0
2		4
3	$\max(8, 4_{(\to 2)} + 5) = \max(8, 9)$	9
4	$\max(9_{(\to 3)} + 1, 4_{(\to 2)} + 7) = \max(10, 11)$	11
5	$\max(4_{(\to 2)} + 8, 11_{(\to 4)} + 2) = \max(12, 13)$	13
6	$\max(13_{(\to 5)} + 6, 11_{(\to 4)} + 5, 9_{(\to 3)} + 6) = \max(19, 16, 15)$	19
7	$\max(13_{(\to 5)} + 7, 19_{(\to 6)} + 2) = \max(20, 21)$	21

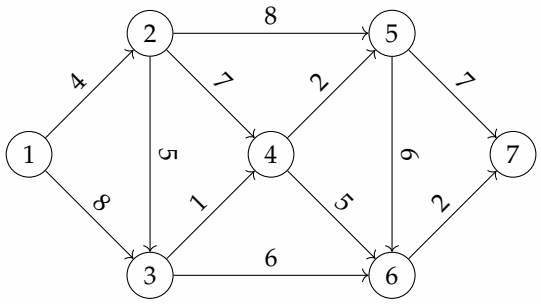
Spätester Termin/Zeitpunkt

i	Nebenrechnung	SZ_i
1	$\min(4_{(\to 2)} - 4, 10_{(\to 3)} - 8) = \min(0, 2)$	0
2	$\min(13_{(\to 5)} - 8, 11_{(\to 4)} - 7, 10_{(\to 3)} - 5) = \min(5, 4, 5)$	4
3	$\min(11_{(\to 4)} - 1, 19_{(\to 6)} - 6) = \min(10, 13)$	10
4	$\min(13_{(\to 5)} - 2, 19_{(\to 6)} - 5) = \min(11, 14)$	11
5	$\min(21_{(\to 7)} - 7, 19_{(\to 6)} - 6) = \min(14, 13)$	13
6	$21_{(\to 7)} - 2$	19
7	siehe FZ_7	21

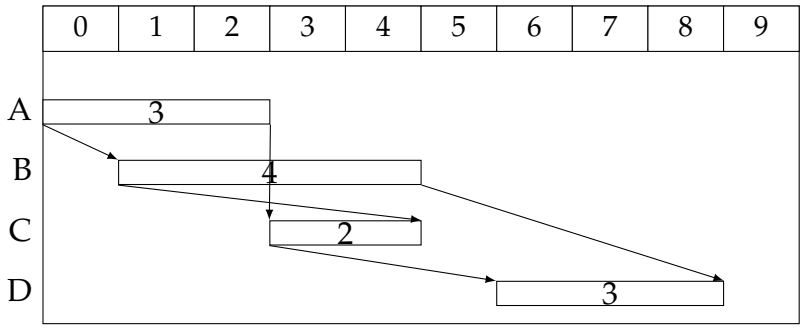
Kritischer Pfad

$1 \to 2 \to 4 \to 5 \to 6 \to 7$

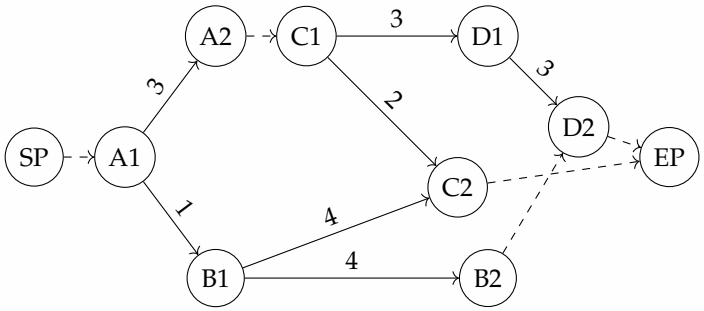
$4_{(1\to 2)} + 7_{(2\to 4)} + 2_{(4\to 5)} + 6_{(5\to 6)} + 2_{(6\to 7)} = 21$



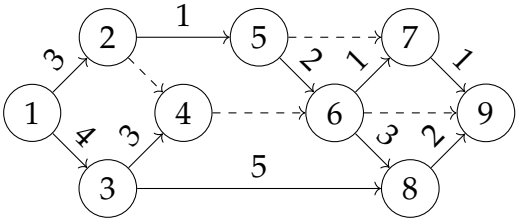
(c) Konvertieren Sie das nachfolgende Gantt-Diagramm in ein CPM-Netzwerk. Als Hilfestellung ist die Anordnung der Ereignisse bereits vorgegeben.



Lösungsvorschlag



66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4



(a) Welche Scheinvorgänge könnten aus dem Netzwerk entfernt werden, ohne dass Informationen verloren gehen?

- ☒ $2 \to 4$
- ☐ $4 \to 6$
- ☒ $5 \to 7$
- ☒ $6 \to 9$

(b) Berechnen Sie für jedes Ereignis den frühesten Termin, wobei angenommen wird, dass das Projekt zum Zeitpunkt 0 startet.

Lösungsvorschlag

i	Nebenrechnung	FZ_i
1		0
2	$0 + 3_{(1\to2)} = 3$	3
3	$0 + 4_{(1\to3)} = 4$	4
4	$3_{(1\to2)} + 0_{(2\to4)} = 3$	
	$4_{(1\to3)} + 3_{(3\to4)} = 7$	
	$\max(3,7)$	7
5	$3_{(1\to2)} + 1_{(2\to5)} = 4$	4
6	$\max(7+0, 4+2)$	7
7	$\max(4+0, 7+1)$	8
8	$\max(4+5, 7+3)$	10
9	$\max(8+1, 7+0, 10+2)$	12

(c) Berechnen Sie für jedes Ereignis auch die spätesten Zeiten, indem Sie für das letzte Ereignis den frühesten Termin als spätesten Termin ansetzen.

Lösungsvorschlag

i	Nebenrechnung	SZ_i
1	$\min(4-3, 4-4)$	0
2	$\min(5-1, 7-0)$	4
3	$\min(10-5, 7-3)$	4
4	$7-0$	7
5	$\min(11-0, 7-2)$	5
6	$\min(12-0, 11-1, 10-3)$	7
7	$12-1$	11
8	$12-2$	10
9	siehe FZ_9	12

(d) Geben Sie nun die Pufferzeiten der Ereignisse an.

Lösungsvorschlag

Ereignis	1	2	3	4	5	6	7	8	9
frühester Termin	0	3	4	7	4	7	8	10	12
spätester Termin	0	4	4	7	5	7	11	10	12
Puffer	0	1	0	0	1	0	3	0	0

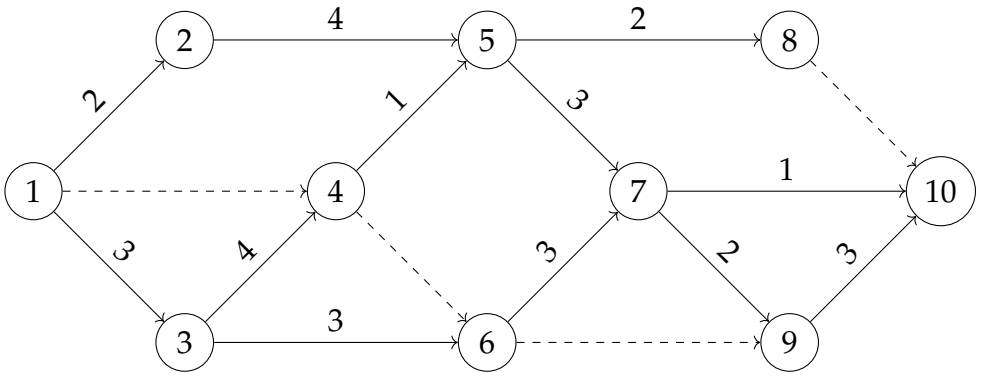
(e) Wie verläuft der kritische Pfad durch das Netzwerk?

Lösungsvorschlag

1 3 4 6 8 9

66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4

Gegeben ist das nachfolgende CPM-Netz. Gestrichelte Linien zwischen Ereignissen stellen Scheinvorgänge mit einer Dauer von 0 dar.



(a) Begründen Sie, welche Scheinvorgänge aus dem Netzplan ohne Informationsverlust gestrichen werden könnten.

Lösungsvorschlag

Die Scheinvorgänge zwischen den Ereignissen 1 und 4 bzw. zwischen 6 und 9 können jeweils gestrichen werden, da Ereignis 4 schon auf 1 wartet (über 3) und 9 wartet auf 6 (über 7).

(b) Berechnen Sie für jedes Ereignis den *frühesten Termin*, den *spätesten Termin* sowie die *Gesamtpufferzeiten*.

Lösungsvorschlag

i	Nebenrechnung	FZ_i
1		0
2		2
3		3
4		7
5	$\max(3_{(\to 3)} + 3, 7_{(\to 4)} + 1)$	8
6	$\max(3_{(\to 3)} + 3, 7_{(\to 4)} + 0)$	7
7	$\max(8_{(\to 5)} + 3, 7_{(\to 6)} + 3)$	11
8	$8_{(\to 5)} + 2$	10
9	$\max(7_{(\to 6)} + 0, 11_{(\to 7)} + 2)$	13
10	$\max(10_{(\to 7)} + 1, 8_{(\to 8)} + 0, 13_{(\to 9)} + 3)$	16

i	Nebenrechnung	SZ_i
1		0
2	$\min(8_{(\to 5)} - 4$	4
3	$\min(8_{(\to 6)} - 3, 7_{(\to 4)} - 4)$	3
4	$\min(8_{(\to 5)} - 1, 8_{(\to 6)} - 0)$	7
5	$\min(16_{(\to 8)} - 2, 11_{(\to 7)} - 3)$	8
6	$\min(11_{(\to 7)} - 3, 13_{(\to 9)} - 0)$	8
7	$\min(16_{(\to 10)} - 1, 13_{(\to 9)} - 2)$	11
8	$16_{(\to 10)} - 0$	16
9	$16_{(\to 10)} - 3$	13
10	siehe $FZ_1 0$	16

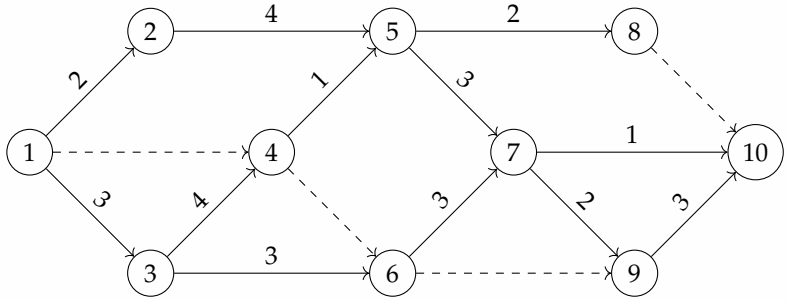
i	1	2	3	4	5	6	7	8	9	10
FZ_i	0	2	3	7	8	7	11	10	13	16
SZ_i	0	4	3	7	8	8	11	16	13	16
GP	0	2	0	0	0	1	0	6	0	0

(c) Bestimmen Sie den kritischen Pfad.

Lösungsvorschlag

$1 \to 3 \to 4 \to 5 \to 7 \to 9 \to 10$

Gantt-Diagramm



66116 / 2021 / Frühjahr / Thema 2 / Aufgabe 4

In Gantt-Diagrammen unterscheidet man vier Anordnungsbeziehungen: Normalfolge (EA), Anfangsfolge (AA), Endfolge (EE) und Sprungfolge (AE). Ordnen Sie folgende Beispielen den Anordnungsbeziehungen (EA, AA, EE, AE) zu.

(a) Tests durchführen und Dokumentation erstellen

Lösungsvorschlag

EE (Das Testen muss abgeschlossen sein bevor die Erstellung der Dokumentation beendet werden kann.)

(b) Systementwurf und Implementierung

Lösungsvorschlag

AA (Der Beginn des Systementwurfs ist Voraussetzung für den Beginn der Implementierung.)

(c) neue Anwendung in Betrieb nehmen und alte Anwendung abschalten

Lösungsvorschlag

AE (Die alte Anwendung kann erst abgeschaltet werden, wenn die neue Anwendung in Betrieb genommen wurde.)

(d) Implementierung und Dokumentation

Lösungsvorschlag

EE (Die Implementierung muss abgeschlossen sein bevor die Dokumentation beendet werden kann.)

(e) Abitur schreiben und Studieren

Lösungsvorschlag

EA (Das Abitur muss abgeschlossen sein bevor mit dem Studium begonnen werden kann.)

(f) Führerschein machen und selbstständiges Autofahren

Lösungsvorschlag CPM-Netzplantechnik

EA (Der Führerschein muss bestanden sein bevor mit dem selbstständigen Autofahren begonnen werden kann.)

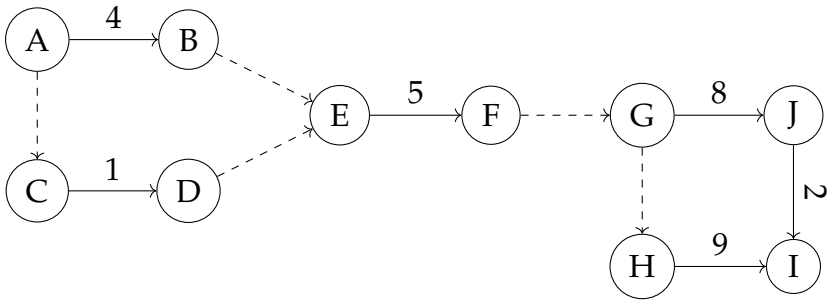
(g) Studieren und Buch aus Uni-Bibliothek ausleihen

Lösungsvorschlag

AA (Der Beginn des Studiums ist Voraussetzung für das Ausleihen eines Buches aus der Uni-Bibliothek.)

46116 / 2015 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 3

Betrachten Sie folgendes CPM-Netzwerk:



(a) Berechnen Sie die früheste Zeit für jedes Ereignis, wobei angenommen wird, dass das Projekt zum Zeitpunkt 0 startet.

Lösungsvorschlag

i	Nebenrechnung	FZ_i
A		0
B		4
C		0
D		1
E	$\max(4,1)$	4
F		9
G		9
H		9
J		17
I	$\max(9_{(\to H)}+9, 17_{(\to J)}+2)$	19

(b) Setzen Sie anschließend beim letzten Ereignis die späteste Zeit gleich der frühesten Zeit und berechnen Sie die spätesten Zeiten.

Lösungsvorschlag Gantt-Diagramm
Gantt-Diagramm

i	Nebenrechnung	SZ_i
A	$\min(3,0)$	0
B		4
C		3
D		4
E		4
F		9
G	$\min(10,9)$	9
H		10
J		17
I		19

(c) Berechnen Sie nun für jedes Ereignis die Pufferzeiten.

Lösungsvorschlag

i	A	B	C	D	E	F	G	H	J	I
FZ_i	0	4	0	1	4	9	9	9	17	19
SZ_i	0	4	3	4	4	9	9	10	17	19
GP	0	0	3	3	0	0	0	1	0	0

(d) Bestimmen Sie den kritischen Pfad.

Lösungsvorschlag

$A \to B \to E \to F \to G \to J \to I$

(e) Was ist ein Gantt-Diagramm? Worin unterscheidet es sich vom CPM-Netzwerk?

Lösungsvorschlag

46116 / 2015 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 3

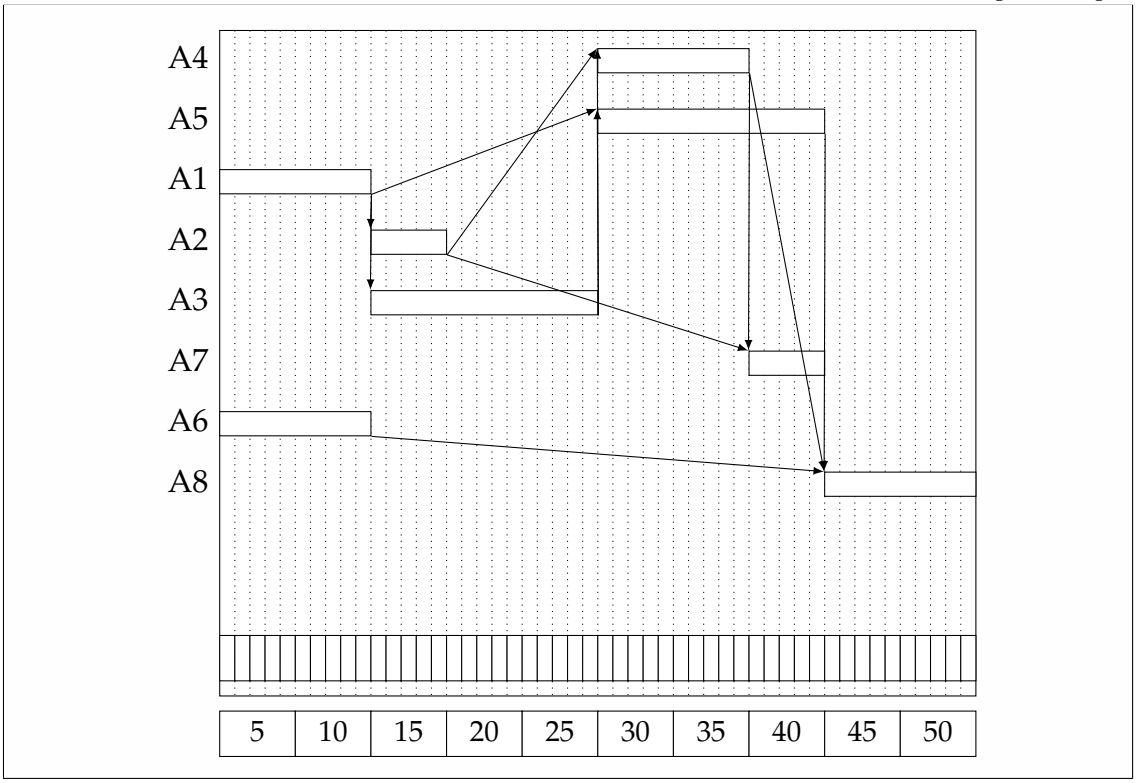
Betrachten Sie die folgende Tabelle zum Projektmanagement:

Name	Dauer (Tage)	Abhängig von
A1	10	
A2	5	A1
A3	15	A1
A4	10	A2, A3
A5	15	A1, A3
A6	10	
A7	5	A2, A4
A8	10	A4, A5, A6

Tabelle 1: Übersicht Arbeitspakete

(a) Erstellen Sie ein Gantt-Diagramm, das die in der Tabelle angegebenen Abhängigkeiten berücksichtigt.

Lösungsvorschlag



(b) Wie lange dauert das Projekt mindestens?

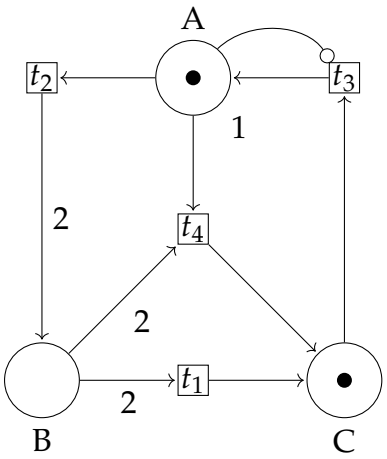
(c) Geben Sie den oder die kritischen Pfad(e) an.

(d) Konstruieren Sie ein PERT-Chart zum obigen Problem.

46116 / 2016 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 2

Petri-Netz
Erreichbarkeitsgraph

Gegeben sei das folgende Petri-Netz:

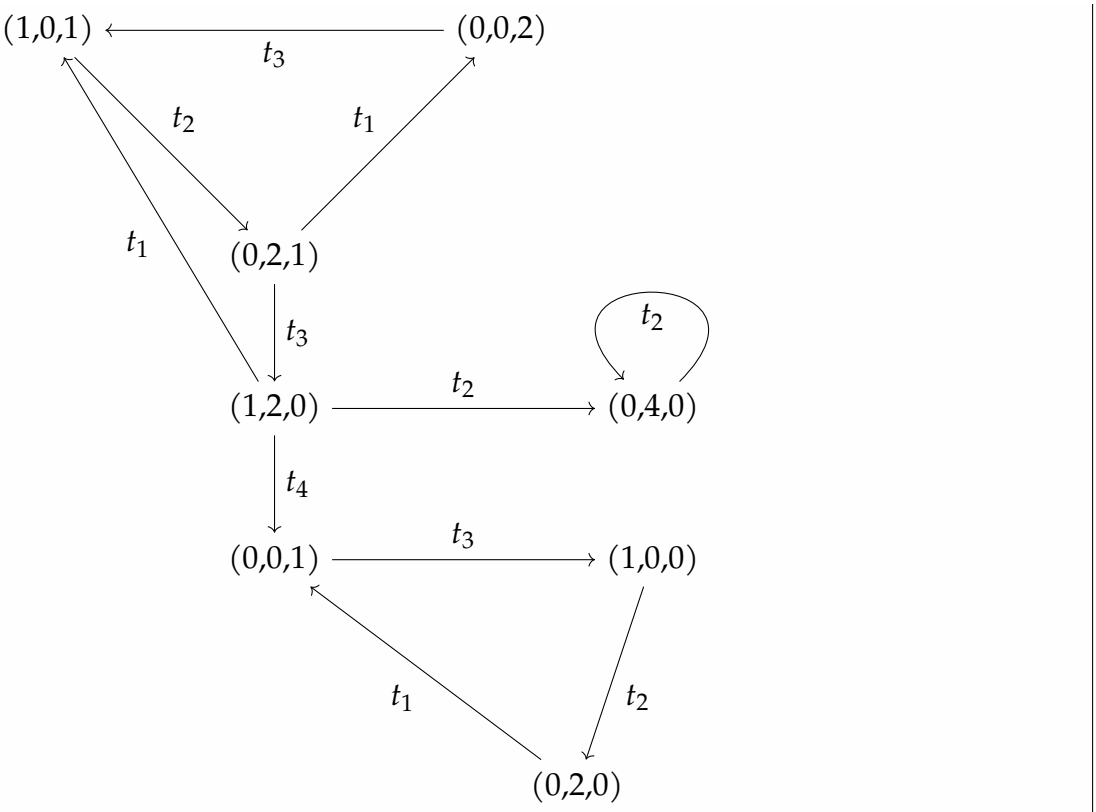


(a) Erstellen Sie den zum Petri-Netz gehörenden Erreichbarkeitsgraphen. Die Belegungen sind jeweils in der Form [A, B, C] anzugeben. Beschriften Sie auch jede Kante mit der zugehörigen Transition. Beachten Sie die auf 1 beschränkte Kapazität von Stelle A oder alternativ die Inhibitor-Kante von A zu t_3 (beides ist hier semantisch äquivalent).

Lösungsvorschlag

$$
\begin{array}{lcll}
(0,0,1) & \rightarrow & t_3 & \rightarrow (1,0,0) \\
(0,0,2) & \rightarrow & t_3 & \rightarrow (1,0,1) \\
(0,2,0) & \rightarrow & t_1 & \rightarrow (0,0,1) \\
(0,2,1) & \rightarrow & t_1 & \rightarrow (0,0,2) \\
(0,2,1) & \rightarrow & t_3 & \rightarrow (1,2,0) \\
(1,0,0) & \rightarrow & t_2 & \rightarrow (0,2,0) \\
(1,0,1) & \rightarrow & t_2 & \rightarrow (0,2,1) \\
(1,2,0) & \rightarrow & t_1 & \rightarrow (1,0,1) \\
(1,2,0) & \rightarrow & t_2 & \rightarrow (0,4,0) \\
(1,2,0) & \rightarrow & t_4 & \rightarrow (0,0,1) \\
(0,4,0) & \rightarrow & t_2 & \rightarrow (0,4,0)
\end{array}
$$

Gantt-Diagramm



(b) Wie kann man mit Hilfe des Erreichbarkeitsgraphen feststellen, ob ein Petri-Netz lebendig ist?

(c) Aufgrund von Transition t_4 ist das gegebene Petri-Netz nicht stark lebendig. Wie müssten die Pfeilgewichte der Transition t_4 verändert werden, damit das Petri-Netz mit der gegebenen Startmarkierung beschränkt bleibt und lebendig wird?

Lösungsvorschlag

t_4 nach C mit Gewicht 2 versehen

46116 / 2017 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 5

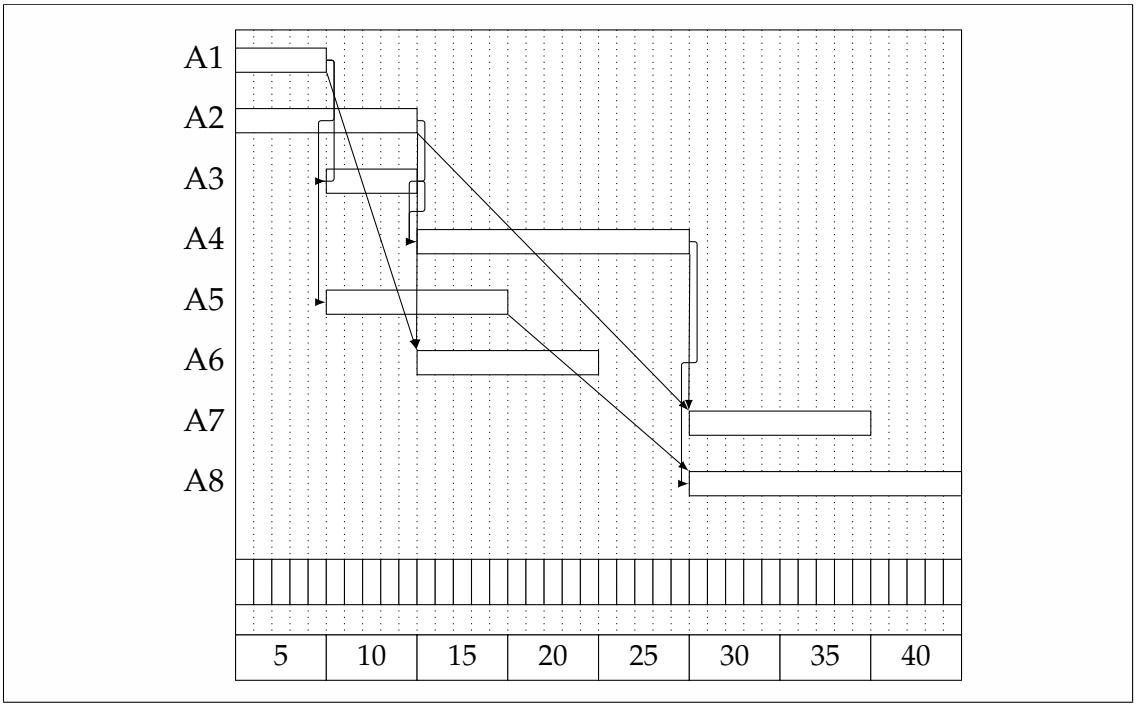
Betrachten Sie die folgende Tabelle zum Projektmanagement:

Name	Dauer (Tage)	Abhängig von
A1	5	
A2	10	
A3	5	A1
AA	15	A2, A3
AS	10	A1
A6	10	A1, A2
A7	10	A2, A4
A8	15	A4, A5

Tabelle 1: Übersicht Arbeitspakete

(a) Erstellen Sie ein Gantt-Diagramm, das die in der Tabelle angegebenen Abhängigkeiten berücksichtigt.

Lösungsvorschlag



(b) Wie lange dauert das Projekt mindestens?

Lösungsvorschlag

40 Tage

(c) Geben Sie den oder die kritischen Pfad(e) an.

Lösungsvorschlag CPM-Netzplantechnik

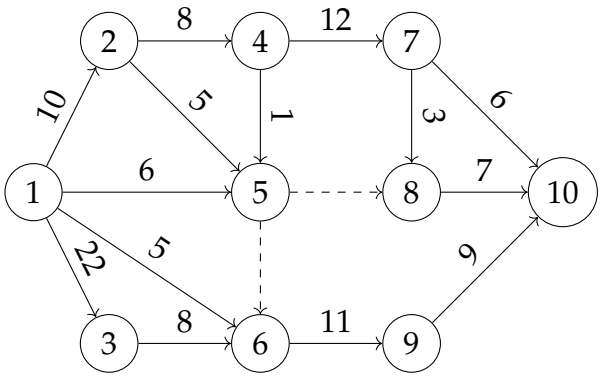
A2 A4 A8

A1 A3 A4 A8

(d) Konstruieren Sie ein PERT-Chart zum obigen Problem.

66116 / 2012 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 2

Die unten stehende Abbildung stellt ein CPM-Netzwerk dar. Die Ereignisse sind fortlaufend nummeriert (Nummer im Inneren der Kreise) und tragen keine Namen. Gestrichelte Linien stellen Pseudo-Aktivitäten mit einer Dauer von 0 dar.



(a) Berechnen Sie die früheste Zeit für jedes Ereignis, wobei angenommen wird, dass das Projekt zum Zeitpunkt 0 startet!

Lösungsvorschlag

— Wir führen eine Vorwärtsterminierung durch und addieren die Dauern. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Maximum aus. **Erläuterungen:** i: Ereignis i; FZ_i: Frühester Zeitpunkt, zu dem Ereignis i eintreten kann. —

i	Nebenrechnung	FZ_i
1		0
2		10
3		22
4		18
5	$\max(15_2, 6_1, 19_4)$	19
6	$\max(5_1, 30_6, 19_5)$	30
7		30
8	$\max(33_7, 19_5)$	33
9		41
10	$\max(36_7, 40_8, 50_9)$	50

(b) Setzen Sie anschließend beim letzten Ereignis die späteste Zeit gleich der frühesten Zeit und berechnen Sie die spätesten Zeiten!

Lösungsvorschlag

— Wir führen eine Rückwärtsterminierung durch und subtrahieren die Dauern vom letzten Ereignis aus. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Minimum aus. **Erläuterungen:** i: Ereignis i; SZ_i: Spätester Zeitpunkt, zu dem Ereignis i eintreten kann. ———

i	Nebenrechnung	SZ_i
10	siehe FZ_{10}	50
9		41
8		43
7	$\min(44_{10}, 40_8)$	40
6		30
5	$\min(30_6, 43_8)$	30
4	$\min(29_5, 28_7)$	28
3		22
2	$\min(20_4, 25_5)$	20
1	$\min(10_2, 24_5, 0_3, 25_6)$	0

(c) Berechnen Sie nun für jedes Ereignis die Pufferzeiten!

Lösungsvorschlag

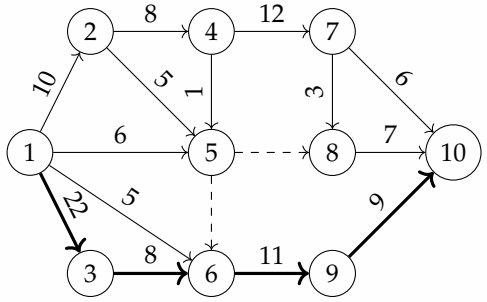
Gantt-Diagramm

i	1	2	3	4	5	6	7	8	9	10
FZ_i	0	10	22	18	19	30	30	33	41	50
SZ_i	0	20	22	28	30	30	40	43	41	50
GP	0	10	0	10	11	0	10	10	0	0

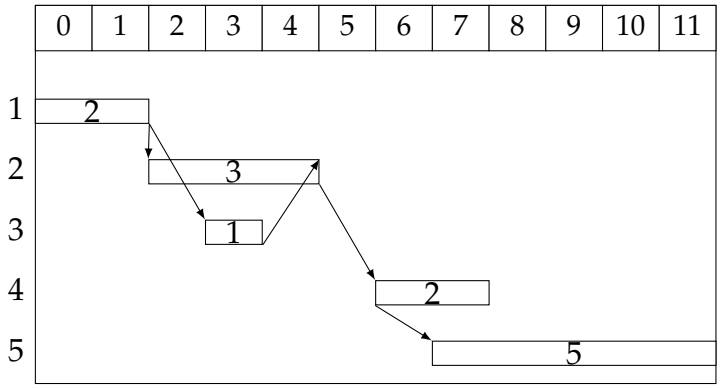
(d) Bestimmen Sie den kritischen Pfad!

Lösungsvorschlag

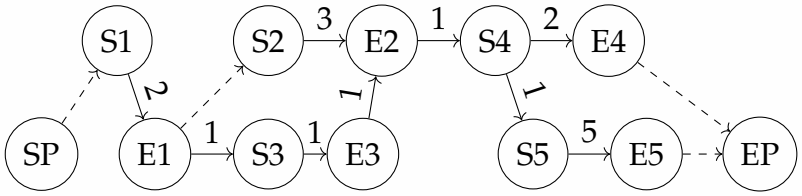
$1 \rightarrow 3 \rightarrow 6 \rightarrow 9 \rightarrow 10$



(e) Konvertieren Sie das Gantt-Diagramm aus Abbildung 3 in ein CPM-Netzwerk!



Lösungsvorschlag



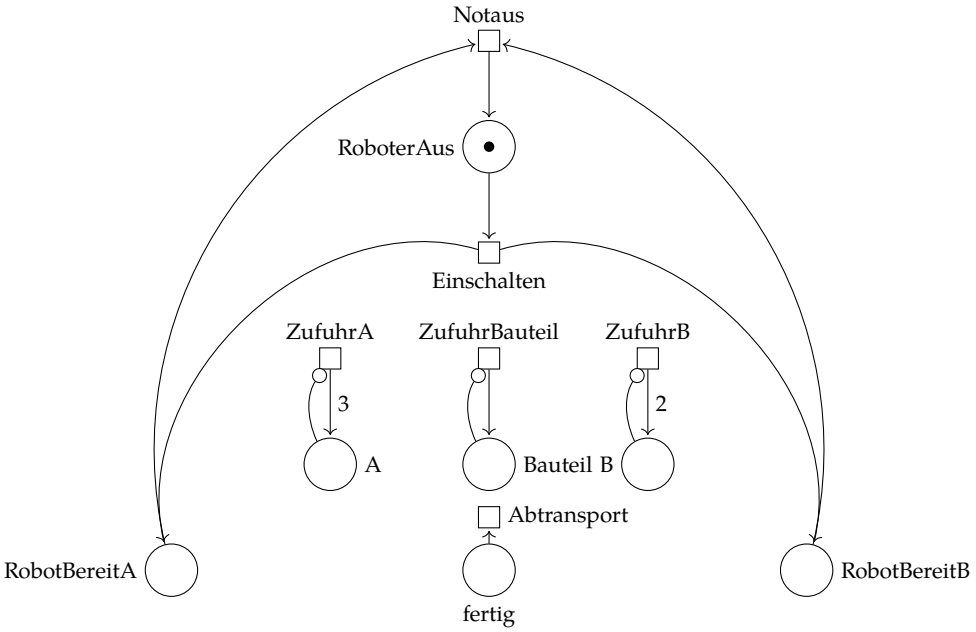
66116 / 2015 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 3

Das folgende Grundgerüst stammt aus dem Petri-Netz-Modell einer Automatisierungsanlage mit zwei Robotern, das Sie auf Ihr Blatt übernehmen und geeignet um weitere

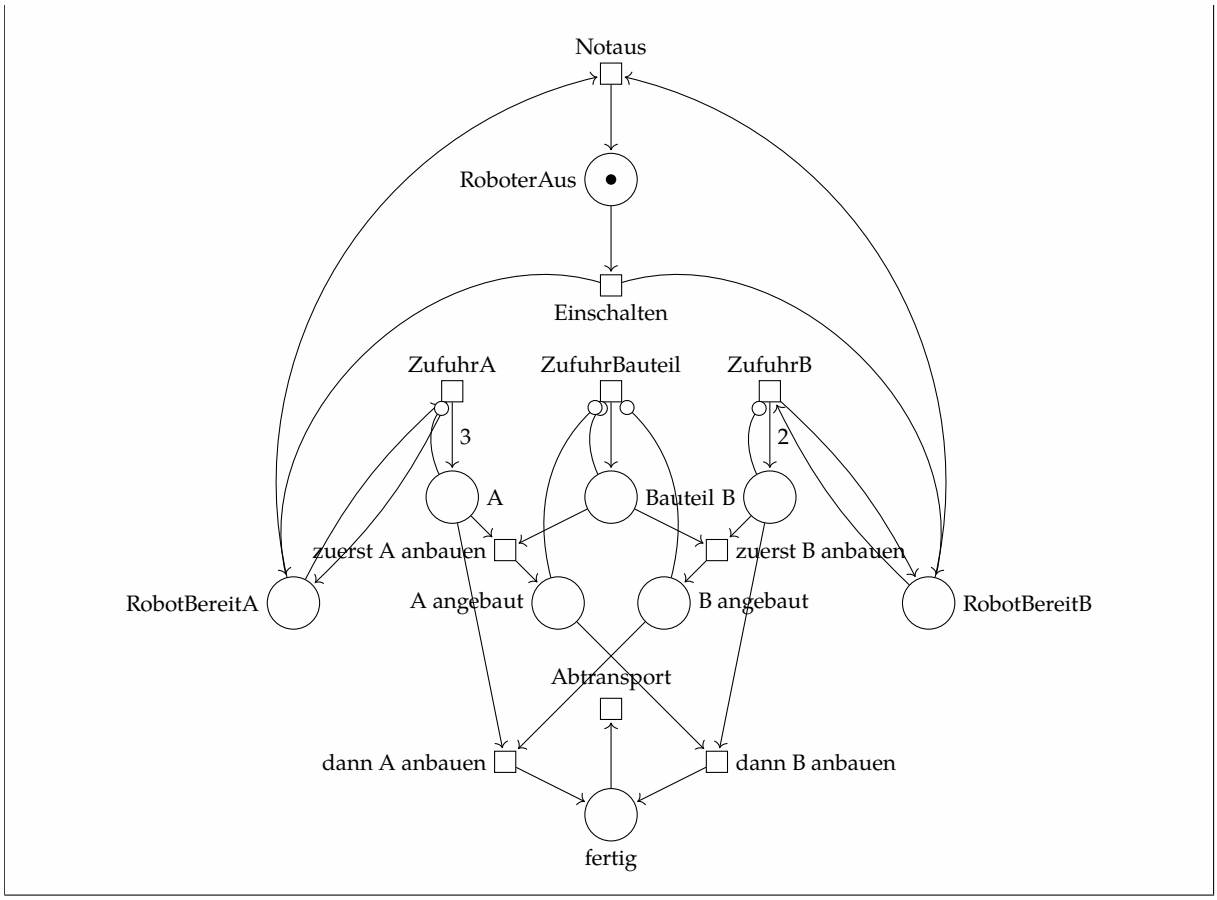
Petri-Netz

Plätze (Stellen), Transitionen, Kapazitäten, Gewichte und Markierungen so ergänzen sollen, dass die darunter angegebenen Anforderungen eingehalten werden:

In der Anlage arbeiten zwei Roboter A und B, die über einen Schalter „Einschalten“ aktiviert und *jederzeit* über einen „Notaus “-Schalter deaktiviert werden können müssen. Aufgabe der Roboter ist es, jeweils abwechselnd an Bauteilen zu arbeiten, die einzeln über ein Förderband „ZufuhrBauteil“ ins System eingefahren und nach der Fertigstellung mittels „Abtransport“ aus dem System abgeführt werden. Roboter A bringt genau 3 Anbauten vom Typ „A“ an jedes Bauteil an und Roboter B macht das entsprechend mit genau 2 Anbauten vom Typ „B“. Die Anbauten werden jeweils passend über „ZufuhrA“ auf „A“ bzw. mittels „ZufuhrB“ auf „B“ bereitgestellt. Die beiden Roboter dürfen *niemals* gleichzeitig an einem Bauteil arbeiten — die Reihenfolge, in der sie darauf zugreifen, ist jedoch beliebig und darf von Bauteil zu Bauteil frei varüeren. Sobald einer der Roboter mit der Arbeit an einem neuen Bauteil begonnen hat, darf kein weiteres Bauteil angenommen werden, ehe der jeweils andere Roboter nicht ebenfalls mit seiner Arbeit am gleichen Bauteil fertig geworden ist (und das fertige Bauteil „auf den Platz ‚fertig‘ legt“). Aus Platzgründen darf höchstens ein Bauteil auf dem Ausgangsplatz „fertig“ abgestellt werden, ehe es abtransportiert wird.



Lösungsvorschlag



66116 / 2016 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 4

(a) Erklären Sie in maximal zwei Sätzen den Unterschied zwischen Knoten- und Kantennetzwerken im Kontext des Projektmanagements.

(b) Gegeben ist die folgende Tabelle zur Grobplanung eines hypothetischen Softwareprojekts:

Aktivität	Minimale Dauer	Einschränkungen
Anforderungsanalyse	2 Monate	Endet frühestens einen Monat nach dem Start der Entwurfsphase.
Entwurf	4 Monate	Startet frühestens zwei Monate nach dem Start der Anforderungsanalyse.
Implementierung	5 Monate	Endet frühestens drei Monate nach dem Ende der Entwurfsphase. Darf erst starten, nachdem die Anforderungsanalyse abgeschlossen ist.

Geben Sie ein CPM-Netzwerkan, das die Aktivitäten und Abhängigkeit des obigen Projektplans beschreibt. Gehen Sie von der Zeiteinheit „Monate“ aus. Das Projekt hat einen Start- und einen Endknoten.

Jede Aktivität wird auf einen Start- und einen Endknoten abgebildet. Die Dauer der Aktivitäten sowie Abhängigkeiten sollen durch Kanten dargestellt werden. Der Start jeder Aktivität hängt vom Projektstart ab, das Projektende hängt vom Ende aller Aktivitäten ab. Modellieren Sie diese Abhängigkeiten durch Pseudoaktivitäten mit Dauer null.

(c) Berechnen Sie für jedes Ereignis (öfür jeden Knoten) die früheste Zeit sowie die späteste Zeit. Beachten Sie, dass die Berechnungsreihenfolge einer topologischen Sortierung des Netzwerks entsprechensollte.

(d) Geben Sie einen kritischen Pfad durch das CPM-Netzwerk an. Möglicherweise sind hierfür weitere Vorberechnungen vonnöten. Welche Aktivität sollte sich demnach auf keinen Fall verzögern?

(e) Geben Sie ein Gantt-Diagramm an, das den Projektplan visualisiert. Gehen Sie davon aus, dass jede Aktivität zur frühesten Zeitihres Startknotens beginnt und zur spätesten Zeit ihres Endknotens endet(s. jeweils Teilaufgabe (c)). Geben Sie die minimale Dauer jeder Aktivität, sowie die Pufferzeit (in Klammern) an. Beispiel: 4 (+2). Notieren Sie alle Einschränkungen mit Hilfe geeigneter Abhängigkeitsbeziehungen. Geben Sie eine absolute Zeitskala in Monaten an.

(f) Nennen Sie zwei weitere Aktivitäten, die in der obigen Tabelle fehlen, jedoch typischerweise in Softwareentwicklungs-Prozessmodellen wie etwa dem Wasserfallmodell vorkommen.

66116 / 2017 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 1

Gegeben ist das folgende Gantt-Diagramm zur Planung eines hypothetischen Softwareprojekts:

(a) Konvertieren Sie das Gantt-Diagramm in ein CPM-Netzwerk, das die Aktivitäten und Abhängigkeiten äquivalent beschreibt. Gehen Sie von der Zeiteinheit „Monate“ aus. Definieren Sie im CPM-Netzwerk je einen globalen Start- und Endknoten. Der Start jeder Aktivität hängt dabei vom Projektstart ab, das Projektende hängt vom Ende aller Aktivitäten ab.

(b) Berechnen Sie für jedes Ereignis (öfür jeden Knoten Ihres CPM-Netzwerks) die früheste Zeit, die späteste Zeit sowie die Pufferzeit. Beachten Sie, dass die Berechnungsreihenfolge einer topologischen Sortierung des Netzwerks entsprechen sollte.

(c) Geben Sie einen kritischen Pfad durch das CPM-Netzwerk an. Welche Aktivität darf sich demnach wie lange verzögern?

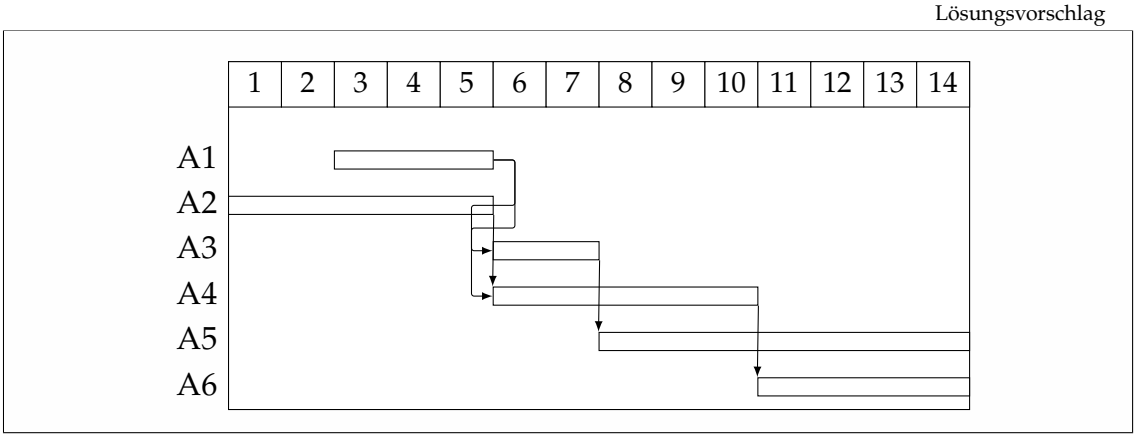
66116 / 2018 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 1

Ein Team von zwei Softwareentwicklern soll ein Projekt umsetzen, das in sechs Arbeitspakete unterteilt ist. Die Dauer der Arbeitspakete und ihre Abhängigkeiten können Sie aus folgender Tabelle entnehmen:

Name	Dauer in Wochen	Abhängig von
A1	2	-
A2	5	-
A3	2	A1
A4	5	A1, A2
A5	7	A3
A6	4	A4

(a) Zeichnen Sie ein Gantt-Diagramm, das eine kürzestmögliche Projektabwicklung beinhaltet.

Lösungsvorschlag



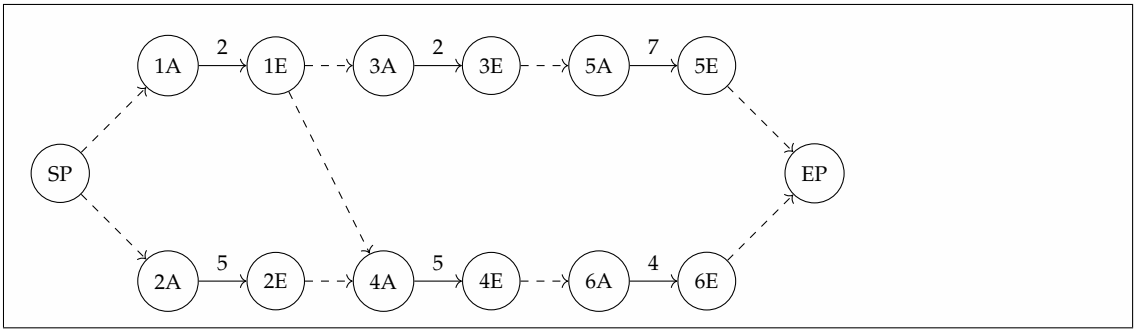
(b) Bestimmen Sie die Länge des kritischen Pfades und geben Sie an, welche Arbeitspakete an ihm beteiligt sind.

Lösungsvorschlag

Auf dem kritischen Pfad befinden die Arbeitspakete A2, A4 und A6. Die Länge des kritischen Pfades ist 14.

(c) Wandeln Sie das Gantt-Diagramm in ein CPM-Netzplan um.

Lösungsvorschlag



(d) Berechnen Sie für jedes Ereignis den *frühesten Termin* und den *spätesten Termin* sowie die *Gesamtpufferzeiten*.

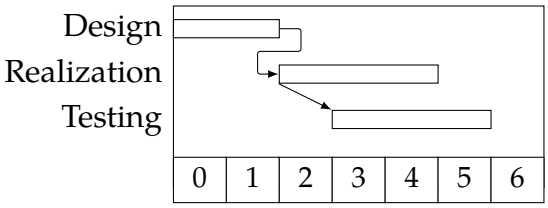
Lösungsvorschlag

i	SP	1A	1E	2A	2E	3A	3E	4A	4E	5A	5E	6A	6E	EP
FZ_i	0	0	2	0	5	2	4	5	10	4	11	10	14	14
SZ_i	0	3	5	0	5	5	7	5	10	7	14	10	14	14
GP	0	3	3	0	0	3	3	0	0	3	3	0	0	0

66116 / 2020 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 2

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?

Lösungsvorschlag

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.

Lösungsvorschlag

D_A Design Anfang

R_A Realization Anfang

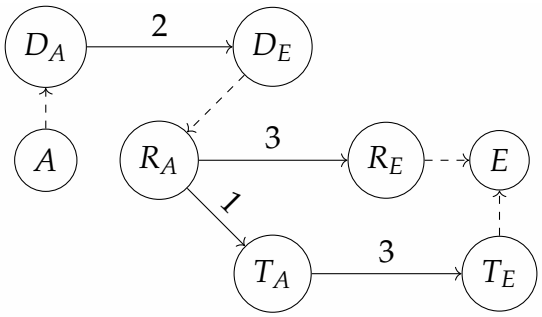
T_A Testing Anfang

D_E Design Ende

R_E Realization Ende

T_E Testing Ende

Gantt-Diagramm



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

Lösungsvorschlag

Diese Beziehungsarten sind im obigen Gantt-Diagramm vorhanden:

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

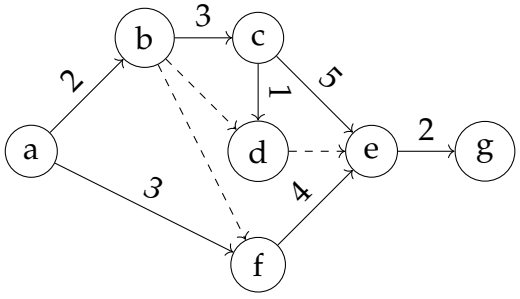
Anfangsfolge AA: *start-to-start relationship* Anordnungsbeziehung vom Anfang eines Vorgangs zum Anfang seines Nachfolgers.

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.

Lösungsvorschlag

— Wir führen eine Vorwärtsterminierung durch und addieren die Dauern. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Maximum aus. **Erläuterungen:** i: Ereignis i; FZ_i: Frühester Zeitpunkt, zu dem Ereignis i eintreten kann. ———————————————————

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.

Lösungsvorschlag

— Wir führen eine Rückwärtsterminierung durch und subtrahieren die Dauern vom letzten Ereignis aus. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Minimum aus. **Erläuterungen:** i: Ereignis i; SZ_i: Spätester Zeitpunkt, zu dem Ereignis i eintreten kann. ———————

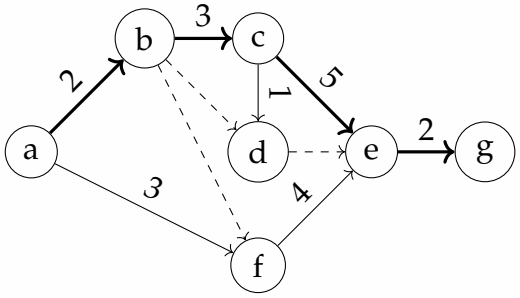
i	Nebenrechnung	SZ_i
g		12
f		6
e		10
d		10
c	$\min(9_d, 5_e)$	5
b	$\min(2_c, 10_d, 6_f)$	2
a		0

(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?

Lösungsvorschlag

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

Gantt-Diagramm



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

Das Projekt verlängert sich um 5 Zeiteinheiten.

66116 / 2020 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 2

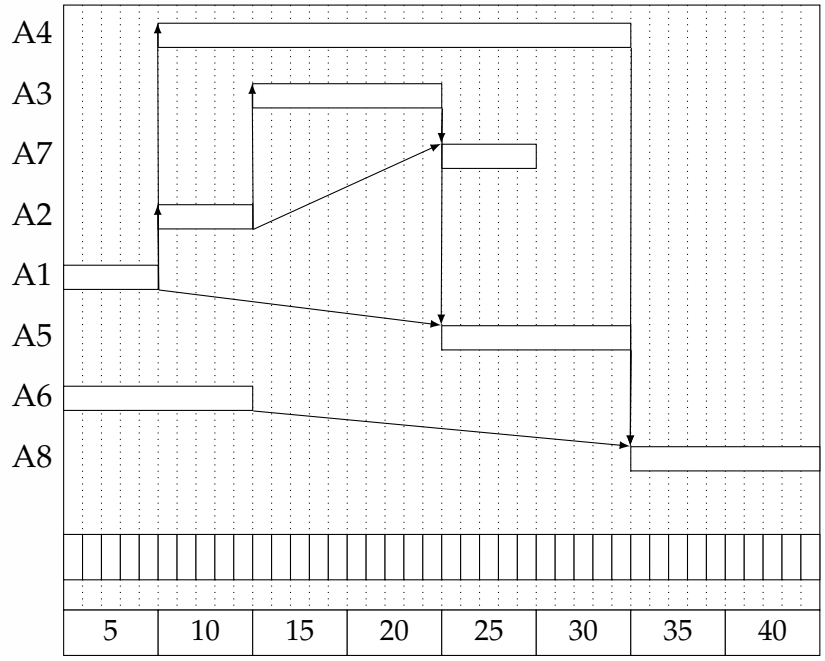
Betrachten Sie die folgende Tabelle zum Projektmanagement:

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.

Lösungsvorschlag

CPM-Netzplantechnik



(b) Wie lange dauert das Projekt mindestens?

Lösungsvorschlag

Es dauert mindestens 40 Tage.

(c) Geben Sie alle kritischen Pfade an.

Lösungsvorschlag

$A1 \rightarrow A4 \rightarrow A8$

$A1 \rightarrow A2 \rightarrow A3 \rightarrow A5 \rightarrow A8$

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

Lösungsvorschlag

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.

66116 / 2021 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 1

Gegeben seien folgende Tätigkeiten mit ihren Abhängigkeiten und Dauern:

Task	Dauer (in h)	Abhängigkeiten
T1	3	/
T2	6	/
T3	2	T1
T4	2	T2
T5	5	T1
T6	3	T4, T5
T7	6	T3
T8	7	T4
T9	4	T6, T8
T10	1	T7, T9

(a) Zeichnen Sie ein CPM-Diagramm basierend auf der gegebenen Aufgabenliste. Benutzen Sie explizite Start- und Endknoten.

Lösungsvorschlag

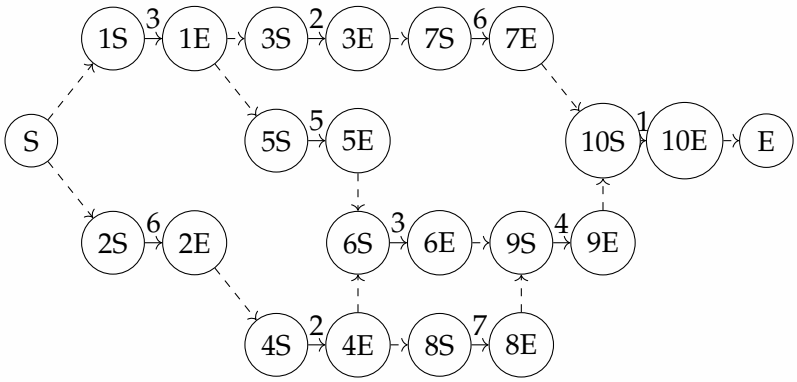
Abkürzungen

S Start

1S Start von T1

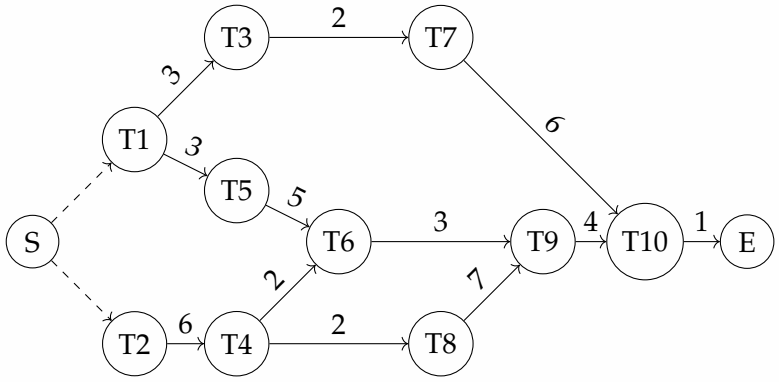
1E Ende von T1

E Ende

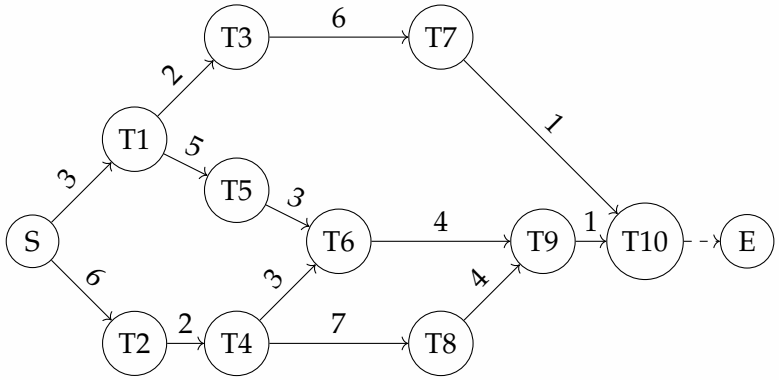


Teilen wir einen Task in zwei Knoten auf, so wird das Diagramm sehr unübersichtlich. Wir verwenden pro Task nur einen Knoten. Es gibt zwei Möglichkeiten:

Knoten sind Anfang der Tasks



Knoten sind Ende der Tasks



(b) Als *Slack* bezeichnet man die Zeit, um die eine Aufgabe bezüglich ihres frühesten Startzeitpunktes verzögert werden kann, ohne dass es Probleme bei der fristgerechten Fertigstellung des Projektes gibt. Berechnen Sie den Slack für alle Aktivitäten und ergänzen Sie ihn in Ihrem Diagramm.

Lösungsvorschlag

Knoten sind Anfang der Tasks

— Wir führen eine Vorwärtsterminierung durch und addieren die Dauern. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Maximum aus. **Erläuterungen:** i: Ereignis i; FZ_i: Frühester Zeitpunkt, zu dem Ereignis i eintreten kann. —

i	Nebenrechnung	FZ_i
T1		0
T2		0
T3		3
T4		6
T5		3
T6	$\max(8_{T4}, 8_{T5})$	8
T7		5
T8		8
T9	$\max(11_{T6}, 15_{T4})$	15
T10	$\max(19_{T9}, 11_{T7})$	19
E		20

— Wir führen eine Rückwärtsterminierung durch und subtrahieren die Dauern vom letzten Ereignis aus. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Minimum aus. **Erläuterungen:** i: Ereignis i; SZ_i: Spätester Zeitpunkt, zu dem Ereignis i eintreten kann. ———————

i	Nebenrechnung	SZ_i
E		20
T10		19
T9		15
T8		8
T7		13
T6		12
T5		7
T4	$\min(12_{T6}, 6_{T8})$	6
T3		11
T2		0
T1	$\min(8_{T3}, 4_{T5})$	4

i	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	E
FZ_i	0	0	3	6	3	8	5	8	15	19	20
SZ_i	4	0	11	6	7	12	13	8	15	19	20
GP	4	0	8	0	4	4	8	0	0	0	0

Knoten sind Ende der Tasks

— Wir führen eine Vorwärtsterminierung durch und addieren die Dauern. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Maximum aus. **Erläuterungen:** i: Ereignis i; FZ_i: Frühester Zeitpunkt, zu dem Ereignis i eintreten kann. ———

i	Nebenrechnung	FZ_i
T1		3
T2		6
T3		5
T4		8
T5		8
T6	$\max(11_{T4}, 11_{T5})$	11
T7		11
T8		15
T9	$\max(15_{T6}, 19_{T8})$	19
T10	$\max(20_{T9}, 12_{T7})$	20
E		20

— Wir führen eine Rückwärtsterminierung durch und subtrahieren die Dauern vom letzten Ereignis aus. Kann ein Ereignis über mehrere Vorgänge erreicht werden, wählen wir das Minimum aus. **Erläuterungen:** i: Ereignis i; SZ_i: Spätester Zeitpunkt, zu dem Ereignis i eintreten kann. ———

i	Nebenrechnung	SZ_i
E		20
T10		20
T9		19
T8		15
T7		19
T6		15
T5		12
T4	$\min(12_{T6}, 8_{T8})$	8
T3		13
T2		6
T1	$\min(11_{T3}, 7_{T5})$	7

i	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	E
FZ_i	3	6	5	8	8	11	11	15	19	20	20
SZ_i	7	6	13	8	12	15	19	15	19	20	20
GP	4	0	8	0	4	4	8	0	0	0	0

(c) Zeichnen Sie den kritischen Pfad in Ihr Diagramm ein oder geben Sie die Tasks des kritischen Pfades in der folgenden Form an: **Start** ! . . . ! **Ende**. Sollte es mehrere kritische Pfade geben, geben Sie auch diese an. Wie lange ist die Dauer des kritischen Pfades bzw. der kritischen Pfade?

Lösungsvorschlag

Kritischer Pfad: **Start** ! T2 ! T4 ! T8 ! T9 ! T10 ! **Ende**

Dauer: 20 h

Softwarearchitektur

66116 / 2019 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 3

Die Komponentenarchitektur eines Softwaresystems beschreibt die unterschiedlichen Softwarebausteine, deren Schnittstellen und die Abhängigkeiten von Softwarekomponenten wie beispielsweise Klassen. Wir unterscheiden zwischen der Soll- und der Ist-Architektur eines Systems.

(a) Nennen und definieren Sie drei Qualitätsattribute von Software, die durch die Architektur beeinflusst werden und charakterisieren Sie jeweils eine „schlechte“ Architektur, die diese Qualitätsattribute negativ beeinflusst.

Lösungsvorschlag

Skalierbarkeit Definition: Ob die Software auch für große Zugriffslasten konzipiert wurde. Charakterisierung einer schlechten Architektur: Eine Anwendung läuft nur auf einem Server.

Modifizierbarkeit Definition: Ob die Software leicht verändert, erweitert werden kann. Charakterisierung einer schlechten Architektur: Monolithische Architektur, dass kein Laden von Modulen zulässt.

Verfügbarkeit Definition: Ob die Software ausfallsicher ist, im Laufenden Betrieb gewartet werden kann. Charakterisierung einer schlechten Architektur: Ein-Server-Architektur, die mehrmal neugestartet werden muss, um ein Update einzuspielen.

(b) Erläutern Sie, was Information Hiding ist. Wie hängt Information Hiding mit Softwarearchitektur zusammen”? Wie wird Information Hiding auf Klassenebene in Java implementiert? Gibt es Situationen, in denen Information Hiding auch negative Effekte haben kann?

Lösungsvorschlag

Das Prinzip der Trennung von Zuständigkeiten (engl. separation of concerns) sorgt dafür, dass jede Komponente einer Architektur nur für eine einzige Aufgabe zuständig ist. Das Innenleben von Komponenten wird durch Schnittstellen verkapselt, was auf das Prinzip des Verbergens von Informationen (engl. information hiding) zurückgeht.

Java: Durch die Sichbarkeits-Schlüsselwörter: private, protected

Zusätzlicher Overhead durch Schnittestellen API zwischen den Modulen.

(c) Erklären Sie, was Refactoring ist.

Lösungsvorschlag Client-Server-Modell
Client-Server-Modell

Verbesserungen des Code durch bessere Lesbarkeit, Wartbarkeit, Performanz, Sicherheit. Keine neuen Funktionen werden programmiert.

(d) Skizzieren Sie die Kernideen von Scrum inkl. der wesentlichen Prozessschritte, Artefakte und Rollen. Beschreiben Sie dann die Rolle von Ist- und Soll-Architektur in agilen Entwicklungskontexten wie Scrum.

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 10

(a) Was bedeutet die Abkürzung AJAX?

Lösungsvorschlag

Asynchronous JavaScript and XML

(b) Erklären Sie in max. drei Sätzen die grundlegende Funktion von AJAX.

Lösungsvorschlag

Konzept der asynchronen Datenübertragung zwischen einem Browser und dem Server. Dieses ermöglicht es, HTTP-Anfragen durchzuführen, während eine HTML-Seite angezeigt wird, und die Seite zu verändern, ohne sie komplett neu zu laden. [a]

[a] `https://de.wikipedia.org/wiki/Ajax_(Programmierung)`

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 11

(a) Was ist das Hypertext Transfer Protocol (HTTP) und wozu dient es?

Lösungsvorschlag

Das Hypertext Transfer Protocol ist ein zustandsloses Protokoll zur Übertragung von Daten auf der Anwendungsschicht über ein Rechnernetz. Es wird hauptsächlich eingesetzt, um Webseiten (Hypertext-Dokumente) aus dem World Wide Web (WWW) in einen Webbrowser zu laden. Es ist jedoch nicht prinzipiell darauf beschränkt und auch als allgemeines Dateiübertragungsprotokoll sehr verbreitet. [a]

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

(b) Betrachten Sie die folgende Zeile Text. Um welche Art von Text handelt es sich?

`https://developer.mozilla.org/en-US/search?q=client+servertoverview`

Lösungsvorschlag

Es handelt sich um eine HTTP-URL (Uniform Resource Locator). Die URL lokalisiert eine Ressource, beispielsweise eine Webseite, über die zu verwendende Zugriffsmethode (zum Beispiel das verwendete Netzwerkprotokoll wie HTTP oder FTP) und den Ort (engl. location) der Ressource in Computer-

netzwerken. [a]

[a] https://de.wikipedia.org/wiki/Uniform_Resource_Locator

Peer-to-Peer-Architektur

(c) Was sind die vier wesentlichen Bestandteile des Texts aus der vorigen Teilaufgabe?

Lösungsvorschlag

Schema https://

Host developer.mozilla.org

Pfad /en-US/search

Query ?q=client+servertoverview

[a]

[a] https://de.wikipedia.org/wiki/Uniform_Resource_Locator

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 12

Es gibt Softwaresysteme, welche auf peer-to-peer (P2P) Kommunikation basieren und eine entsprechende Architektur aufweisen.

(a) Bewerten Sie die folgenden Aussagen als entweder richtig oder falsch.

(i) Mithilfe des Befehls "lookup" können Peers sich gegenseitig identifizieren.

Lösungsvorschlag

richtig

(ii) In einem P2P-System, wie auch bei Client-Server, sind alle Netzwerkteilnehmer gleichberechtigt.

Lösungsvorschlag

falsch. Im Client-Servermodell sind nicht alle Netzwerkteilnehmer gleichberechtig. Der Server hat mehr Privilegien wie der Client.

(iii) Alle P2P-Systeme funktionieren grundsätzlich ohne einen zentralen Verwaltungs-Peer.

Lösungsvorschlag

falsch. Es gibt zentralisierte P2P-Systeme (Beispiel: Napster), welche einen zentralen Server zur Verwaltung benötigen, um zu funktionieren. [a]

[a] https://de.wikipedia.org/wiki/Peer-to-Peer

(iv) P2P kann auch für eine Rechner-Rechner-Verbindung stehen.

Lösungsvorschlag Client-Server-Modell

richtig

(v) Es gibt strukturierte und unstrukturierte P2P-Systeme. In unstrukturierten P2P-Systemen wird zum Auffinden von Peers eine verteilte Hashtabelle verwendet (DHT).

Lösungsvorschlag

richtig

(vi) In einem P2P-System sind theoretisch alle Peers gleichberechtigt, praktisch gibt es jedoch leistungsabhängige Gruppierungen.

Lösungsvorschlag

richtig

(vii) Ein Peer kann sowohl ein Client wie auch ein Server für einen anderen Peer sein.

Lösungsvorschlag

richtig

(b) Wählen Sie zwei falsche Aussagen aus der vorherigen Tabelle aus und berichtigen Sie diese in jeweils einem Satz.

Lösungsvorschlag

Sie oben.

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 8

Das Client-Server-Modell ist ein Architekturmuster. Nennen Sie zwei Vorteile einer nach diesem Muster gestalteten Architektur.

Lösungsvorschlag

(a) Einfache Integration weiterer Clients

(b) Prinzipiell uneingeschränkte Anzahl der Clients [a]

(c) Es muss nur ein Server gewartet werden. Dies gilt z. B. für Updates, die einmalig und zentral auf dem Server durchgeführt werden und danach für alle Clients verfügbar sind. [b]

[a] https://www.karteikarte.com/card/164928/vorteile-und-nachteile-des-client-server-modells
[b] https://www.eoda.de/wissen/blog/client-server-architekturen-performance-und-agilitaet-fuer-data-s

66116 / 2021 / Frühjahr / Thema 1 / Teilaufgabe 1 / Aufgabe 9

Client-Server-Modell

Betrachten Sie die folgende Liste von Technologien:

- Nodejs
- PHP
- CSS
- AJAX
- Python
- Java

Welche dieser Technologien laufen in einem Client-Server-System üblicherweise auf der Seite des Klienten und welche auf der Seite des Servers? Nehmen Sie hierzu an, dass der Client ein Browser ist.

Übertragen Sie die im Folgenden gegebene Tabelle in Ihren Bearbeitungsbogen und ordnen Sie die aufgelisteten Technologien anhand der Buchstaben in die Tabelle ein. Fortsetzung nächste Seite!

Hinweis: Mehrfachzuordnungen sind möglich.

Lösungsvorschlag

Client-seitige Technologien	Server-seitige Technologien
CSS	Nodejs
AJAX	PHP
Python	Python
Java	Java

Testen

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

Gegeben sei folgende rekursive Methodendeklaration in der Sprache Java. Es wird als Vorbedingung vorausgesetzt, dass die Methode `sum` nur für Werte $n \geq 0$ aufgerufen wird.

```java
public static int sum(int n) {
  if (n <= 0) {
    return 0;
  }
  return n + sum(n - 1);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/induktion/Gauss.java`

Beweisen Sie mittels vollständiger Induktion, dass der Methodenaufruf `sum(n)` die Summe der ersten n aufeinanderfolgenden natürlichen Zahlen für alle $n \geq 0$ berechnet, wobei gilt

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ———————————

$$\sum_{k=0}^{0} k = \frac{0(0+1)}{2} = \frac{0}{2} = 0$$

$$\texttt{sum(0)} = 0$$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ———————————

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

$$\texttt{sum(n)} = n + \frac{(n-1)((n-1)+1)}{2} = n + \frac{(n-1)n}{2}$$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss. —

$$\sum_{k=0}^{n+1} k = \frac{(n+1)\big((n+1)+1\big)}{2}$$

$$\begin{aligned}
\texttt{sum(n+1)} &= (n+1) + \frac{((n+1)-1)(n+1))}{2} && n+1-1 = n \\
&= (n+1) + \frac{n(n+1)}{2} && (n+1) \text{ eingesetzt} \\
&= \frac{2(n+1)}{2} + \frac{n(n+1)}{2} && (n+1) \text{ als Bruch geschrieben} \\
&= \frac{2(n+1)+n(n+1)}{2} && \text{Hauptnenner 2} \\
&= \frac{(2+n)(n+1)}{2} && (n+1) \text{ ausgeklammert} \\
&= \frac{(n+2)(n+1)}{2} && \text{Kommutativgesetz angewandt} \\
&= \frac{(n+1)(n+2)}{2} && \text{getauscht nach Kommutativgesetz} \\
&= \frac{(n+1)\big((n+1)+1\big)}{2} && \text{mit } (n+1) \text{ an der Stelle von } n
\end{aligned}$$

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class GaussTest {

  private void teste(int n, int erwartet) {
    assertEquals(Gauss.sum(n), erwartet);
  }

  @Test
  public void teste() {
    teste(0, 0);
    teste(1, 1);
    teste(2, 3);
    teste(3, 6);
    teste(4, 10);
    teste(5, 15);
    teste(6, 21);
    teste(7, 28);
    teste(8, 36);
    teste(9, 45);
    teste(10, 55);
    teste(11, 66);
  }

}
```

Vollständige Induktion

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/aufgaben/aud/induktion/GaussTest.java`

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

Gegeben sei folgende Methode:

```java
public class GeoSum {
  // Math.pow(q, n) == q^n
  double geoSum(int n, double q) {
    if (n == 0) {
      return 1 - q;
    } else {
      return (1 - q) * Math.pow(q, n) + geoSum(n - 1, q);
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/sosy/totale_korrektheit/GeoSum.java`

Weisen Sie mittels vollständiger Induktion nach, dass

$$\text{geoSum}(n, q) = 1 - q^{n+1}$$

Dabei können Sie davon ausgehen, dass $q > 0$, $n \in \mathbb{N}_0$

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ———————————

$$f(0) : \text{geoSum}(0, q) = 1 - q^{0+1} = 1 - q^1 = 1 - q$$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ———————————

$$f(n) : \text{geoSum}(n, q) = 1 - q^{n+1}$$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss. ——

$$\begin{aligned}
f(n) &= \text{geoSum}(n, q) \\
&= (1 - q) \cdot q^n + \text{geoSum}(n - 1, q) && \text{Java-Code in Mathe-Formel umgewandelt} \\
&= (1 - q) \cdot q^n + 1 - q^{(n-1)+1} && \text{für rekursiven Methodenaufruf gegebene Formel eingesetzt} \\
&= (1 - q) \cdot q^n + 1 - q^n && \text{Addition im Exponent}
\end{aligned}$$

$$
\begin{aligned}
f(n+1) &= \text{geoSum}(n+1, q) \\
&= (1-q) \cdot q^{n+1} + 1 - q^{n+1} && \text{von Java konvertierte Formel verwendet und } n+1 \text{ eingesetzt} \\
&= q^{n+1} - q^{(n+1)+1} + 1 - q^{n+1} && \text{ausmultipliziert} \\
&= -q^{(n+1)+1} + 1 && q^{n+1} - q^{n+1} = 0 \\
&= 1 - q^{(n+1)+1} && \text{Kommutativgesetz der Addition} \\
&= 1 - q^{(n+1)+1} && \text{was zu zeigen war}
\end{aligned}
$$

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

Die schrittweise Berechnung der Summe der ersten n ungeraden Zahlen legt die Vermutung nahe: Die Summe aller ungeraden Zahlen von 1 bis $2n-1$ ist gleich dem Quadrat von n:

$$
\begin{aligned}
1 &= 1 = 1^2 \\
1 + 3 &= 4 = 2^2 \\
1 + 3 + 5 &= 9 = 3^2 \\
1 + 3 + 5 + 7 &= 16 = 4^2
\end{aligned}
$$

Folgende Java-Methode berechnet die Summer aller ungeraden Zahlen:

```java
public static int oddSum(int n) {
  if (n <= 1) {
    return 1;
  }
  return 2 * n - 1 + oddSum(n - 1);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/aud/induktion/Maurolicus.java`

Beweisen Sie mittels vollständiger Induktion, dass der Methodenaufruf `oddSum(n)` die Summe aller ungeraden Zahlen von 1 bis nur n-ten ungeraden Zahl berechnet, wobei gilt:

$$\sum_{i=1}^{n}(2i-1) = n^2$$

.

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist.

$$\sum_{i=1}^{1}(2i-1) = 2 \cdot 1 - 1 = 1 = 1^2$$

$$\texttt{oddSum(1)} = 1 = 1^2$$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ——————————

$$\sum_{i=1}^{n}(2i-1) = n^2$$

$$\texttt{oddSum(n)} = 2n - 1 + (n-1)^2$$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k+1)$ wahr sein muss. ——

$$\begin{aligned}
\texttt{oddSum(n)} &= 2(n+1) - 1 + ((n+1) - 1)^2 \\
&= 2(n+1) - 1 + n^2 \\
&= 2n + 2 + n^2 - 1 && \text{ausmultiplizieren} \\
&= 2n+1 + n^2 && 2 - 1 = 1 \\
&= n^2 + 2n + 1 && \text{Kommutativgesetz} \\
&= (n+1)^2 && \text{mit erster Binomischer Formel: } (a+b)^2 = a^2 + 2ab + b^2
\end{aligned}$$

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class MaurolicusTest {

  private void teste(int n, int erwartet) {
    assertEquals(Maurolicus.oddSum(n), erwartet);
  }

  @Test
  public void teste() {
    teste(1, 1);
    teste(2, 4);
    teste(3, 9);
    teste(4, 16);
    teste(5, 25);
    teste(6, 36);
    teste(7, 49);
    teste(8, 64);
    teste(9, 81);
    teste(10, 100);
    teste(11, 121);
  }
```

```
}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/aufgaben/aud/induktion/MaurolicusTest.java`

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

(a) Geben Sie zwei verschiedene Möglichkeiten der formalen Verifikation an.

Lösungsvorschlag

1. Möglichkeit: formale Verifikation mittels *vollständiger Induktion* (eignet sich bei *rekursiven* Programmen).

2. Möglichkeit: formale Verifikation mittels *wp-Kalkül* oder *Hoare-Kalkül* (eignet sich bei *iterativen* Programmen).

(b) Erläutern Sie den Unterschied von partieller und totaler Korrektheit.

Lösungsvorschlag

partielle Korrektheit: Das Programm verhält sich spezifikationsgemäß, *falls* es terminiert.

totale Korrektheit: Das Programm verhält sich spezifikationsgemäß und es *terminiert immer*.

(c) Gegeben sei die Anweisungssequenz A. Sei P die Vorbedingung und Q die Nachbedingung dieser Sequenz. Erläutern Sie, wie man die (partielle) Korrektheit dieses Programmes nachweisen kann.

Lösungsvorschlag

Vorgehen	Horare-Kalkül	wp-Kalkül
Wenn die Vorbedingung P zutrifft, gilt nach der Ausführung der Anweisungssequenz A die Nachbedingung Q.	$\{P\}A\{Q\}$	$P \Rightarrow \text{wp}(A, Q)$

(d) Gegeben sei nun folgendes Programm:

```
A_1
while(b):
    A_2
A_3
```

wobei A_1, A_2, A_3 Anweisungssequenzen sind. Sei P die Vorbedingung und Q die Nachbedingung des Programms. Die Schleifeninvariante der while-Schleife wird mit I bezeichnet. Erläutern Sie, wie man die (partielle) Korrektheit dieses Programmes nachweisen kann.

Lösungsvorschlag

Terminierungsfunktion
wp-Kalkül

Vorgehen	Horare-Kalkül	wp-Kalkül
Die Invariante I gilt vor Schleifeneintritt.	$\{P\}A_1\{I\}$	$P \Rightarrow \text{wp}(A_1, I)$
I ist invariant, d. h. I gilt nach jedem Schleifendurchlauf.	$\{I \wedge b\}A_2\{I\}$	$I \wedge b \Rightarrow \text{wp}(A_2, I)$
Die Nachbedingung Q wird erfüllt.	$\{I \wedge \neg b\}A_3\{Q\}$	$I \wedge \neg b \Rightarrow \text{wp}(A_3, I)$

(e) Beschreiben Sie, welche Vorraussetzungen eine Terminierungsfunktion erfüllen muss, damit die totale Korrektheit gezeigt werden kann.

Lösungsvorschlag

Mit einer Terminierungsfunktion T kann bewiesen werden, dass eine Wiederholung terminiert. Sie ist eine Funktion, die

- ganzzahlig,
- nach unten beschränkt (die Schleifenbedingung ist *false*, wenn $T = 0$) und
- streng monoton fallend (jede Ausführung der Wiederholung verringert ihren Wert)

ist.

Im Hoare-Kalkül muss $\{I \wedge b \wedge (T = n)\}A\{T < n\}$ gezeigt werden, im wp-Kalkül $I \Rightarrow T \geq 0$. [a]

[a] https://osg.informatik.tu-chemnitz.de/lehre/aup/aup-07-AlgorithmenEntwurf-script_de.pdf

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

Gegeben sei folgendes Programm: wp-Kalkül

```
int f(int x, int y) {
  /* P */
  x = 2 * x + 1 + x * x;
  y += 7;
  if (x > 196) {
    y = 2 * y;
  } else {
    y -= 8;
    x *= 2;
  } /* Q */
  return x + y;
}
```

Bestimmen Sie die schwächste Vorbedingung (weakest precondition), für die die Nachbedingung $Q := (x \geq 8) \wedge (y\%2 = 1)$ noch zutrifft.

Lösungsvorschlag

Mit dem Distributivgesetz der Konjugation gilt:

$$\begin{aligned}
&\text{wp}(\texttt{"A; if(b) B; else C;"}, Q) \equiv \\
&\text{wp}(\texttt{"A;"}, b) \land \text{wp}(\texttt{"A;B;"}, Q) \\
&\lor \\
&\text{wp}(\texttt{"A;"}, \neg b) \land \text{wp}(\texttt{"A;C;"}, Q)
\end{aligned}$$

Der tatsächliche Programmcode wird eingesetzt:

$$\begin{aligned}
&\text{wp}(\texttt{"x=2*x+1+x*x;y+=7;if(x>196){y=2*y;}else{y-=8;x*=2;}"}, (x \geq 8) \land (y\%2 = 1)) \equiv \\
&\text{wp}(\texttt{"x=2*x+1+x*x;y+=7;"}, x > 196) \land \\
&\text{wp}(\texttt{"x=2*x+1+x*x;y+=7;y=2*y;"}, (x \geq 8) \land (y\%2 = 1)) \\
&\lor \\
&\text{wp}(\texttt{"x=2*x+1+x*x;y+=7;"}, x \leq 196) \land \\
&\text{wp}(\texttt{"x=2*x+1+x*x;y+=7;y-=8;x*=2;"}, (x \geq 8) \land (y\%2 = 1)) \\
&=: \text{P}
\end{aligned}$$

Nebenrechnung: $\text{wp}(\texttt{"A;"}, b)$

bWpPseudoMatheUmgebung $\text{wp}(\texttt{"x=2*x+1+x*x;y+=7;"}, x > 196)$

Wir lassen $y+=7$ weg, weil in der Nachbedingung kein y vorkommt und setzen in den Term $x > 196$ für das x die erste Code-Zeile $2 \cdot x + 1 + x \cdot x$ ein.

$$\equiv \quad \text{wp}(\texttt{""}, 2 \cdot x + 1 + x \cdot x > 196)$$

Nach der Transmformationsregel *Nichts passiert, die Vorbedingung bleibt gleich* kann das auch so geschrieben werden:

$$\equiv \quad 2 \cdot x + 1 + x \cdot x > 196$$

Die erste binomische Formel (Plus-Formel) lautet $(a+b)^2 = a^2 + 2ab + b^2$. Man kann die Formel auch umgedreht verwenden: $a^2 + 2ab + b^2 = (a+b)^2$. Die erste Code-Zeile $2 \cdot x + 1 + x \cdot x$ kann umformuliert werden in $1 + 2 \cdot 1 \cdot x + x \cdot x = 1^2 + 2 \cdot 1 \cdot x + x^2 = (1+x)^2 = (x+1)^2$. Wir haben für a die Zahl 1 und für b den Buchstaben x eingesetzt.

$$\equiv \quad (x+1)^2 > 196$$

Nebenrechnung: $\text{wp}(\texttt{"A;B;"}, Q)$

bWpPseudoMatheUmgebung $\text{wp}(\texttt{"x=2*x+1+x*x;y+=7;y=2*y;"}, (x \geq 8) \land (y\%2 = 1))$

Für das x in der Nachbedingung setzen wir die erste Code-Zeile $2 \cdot x + 1 + x \cdot x$ ein. Für das y in der Nachbedingung setzen wir dritte Code-Zeile `y=2*y;` ein und dann die zweite Code-Zeile `y+=7;`. Das wp-Kalkül arbeitet den Code rückswärts ab. in $y\%2$ die dritte Anweisung $y = 2 \cdot y$ einfügen: $2 \cdot y\%2$ dann in $2 \cdot y\%2$ die zweite Anweisung $y = y + 7$ einfügen: $2 \cdot (y+7)\%2$

$$\equiv \quad (x+1)^2 \geq 8 \land 2(y+7)\%2 = 1$$

Diese Aussage ist falsch, da $2(y+7)$ immer eine gerade Zahl ergibt und der Rest von einer Division durch zwei einer geraden Zahl immer 0 ist und nicht 1.

$\equiv \quad (x+1)^2 \geq 8 \wedge \text{falsch}$

$\equiv \quad \text{falsch}$

Nebenrechnung: wp("A;", $\neg b$)

bWpPseudoMatheUmgebung wp("x=2*x+1+x*x;y+=7;", $x \leq 196$)

Analog zu Nebenrechnung 1

$\equiv \quad (x+1)^2 \leq 196$

Nebenrechnung: wp("A;C;", Q)

bWpPseudoMatheUmgebung wp("x=2*x+1+x*x;y+=7;y-=8;x*=2;", $(x \geq 8) \wedge (y\%2 = 1)$)

„x*=2;“: $x \cdot 2$ für x einsetzen:

$\equiv \quad \text{wp}(\text{"x=2*x+1+x*x;y+=7;y-=8;"}, (2 \cdot x \geq 8) \wedge (y\%2 = 1))$

„y-=8;“: $y - 8$ für y einsetzen:

$\equiv \quad \text{wp}(\text{"x=2*x+1+x*x;y+=7;"}, (2 \cdot x \geq 8) \wedge ((y-8)\%2 = 1))$

„y+=7“: $y + 7$ für y einsetzen:

$\equiv \quad \text{wp}(\text{"x=2*x+1+x*x;"}, (2 \cdot x \geq 8) \wedge (((y+7)-8)\%2 = 1))$

„x=2*x+1+x*x;“: $(x+1)^2$ für x einsetzen:

$\equiv \quad \text{wp}(\text{""}, (2 \cdot (x+1)^2 \geq 8) \wedge (((y+7)-8)\%2 = 1))$

Nur noch die Nachbedingung stehen lassen:

$\equiv \quad (2 \cdot (x+1)^2 \geq 8) \wedge (((y+7)-8)\%2 = 1)$

Subtraktion:

$\equiv \quad (2 \cdot (x+1)^2 \geq 8) \wedge ((y-1)\%2 = 1)$

Vereinfachen (links beide Seiten durch 2 teilen und rechts von beiden Seiten 1 abziehen)

$\equiv \quad (\frac{2 \cdot (x+1)^2}{2} \geq \frac{8}{2}) \wedge (((y-1)\%2) - 1 = 1 - 1)$

Zwischenergebnis:

$\equiv \quad ((x+1)^2 \geq 4) \wedge y\%2 = 0$

Zusammenführung

Die Zwischenergebnisse aus den Nebenrechnungen zusammenfügen:

$\equiv \quad [(x+1)^2 > 196 \wedge \text{falsch}] \vee [(x+1)^2 \leq 196 \wedge (x+1)^2 \geq 4 \wedge y\%2 = 0]$

„falsch“ und eine Aussage verbunden mit logischem Und „$\wedge$“ ist insgesamt falsch:

wp-Kalkül

$$\equiv \quad \text{falsch} \vee [(x+1)^2 \leq 196 \wedge (x+1)^2 \geq 4 \wedge y\%2 = 0]$$

falsch verbunden mit oder weglassen:

$$\equiv \quad (x+1)^2 \leq 196 \wedge (x+1)^2 \geq 4 \wedge y\%2 = 0$$

Umgruppieren, sodass nur noch ein $(x+1)^2$ geschrieben werden muss:

$$\equiv \quad 4 \leq (x+1)^2 \leq 196 \wedge y\%2 = 0$$

$4 = 2^2$ und $196 = 14^2$

$$\equiv \quad 2^2 \leq (x+1)^2 \leq 14^2 \wedge y\%2 = 0$$

Hoch zwei weg lassen: Betragsklammer $|x|$ oder auch Betragsfunktion hinzufügen (Die Betragsfunktion ist festgelegt als „Abstand einer Zahl von der Zahl Null“.

$$\equiv \quad 2 \leq |x+1| \leq 14 \wedge y\%2 = 0$$

Auf die Gleichung der linken Aussage -1 anwenden:

$$\equiv \quad 1 \leq |x| \leq 13 \wedge y\%2 = 0$$

Die Betragsklammer weg lassen:

$$\equiv \quad (1 \leq x \leq 13 \vee -13 \leq x \leq -1) \wedge y\%2 = 0$$

bWpPseudoMatheUmgebung=: P

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

Bestimmen Sie zur Nachbedingung Q die Vorbedingung P!

Nachbedingung: $Q \equiv x + y = 17$

Programmcode:

```
// P: ?
x += 5;
y *= 2;
z = z % 4;
y--;
// Q: x + y = 17
```

Lösungsvorschlag

ist gleichbedeutet mit

```
x = x + 5;
y = y * 2;
z = z % 4;
y = y - 1;
```

$$\text{wp}(\texttt{"x += 5; y *= 2; z = z \% 4; y--;"}, x + y = 17)$$

$y - 1$ einsetzten

$$\equiv \quad \text{wp}(\texttt{"x += 5; y *= 2; z = z \% 4;"}, x + y - 1 = 17)$$

die 1 mit + nach rechts bringen

$$\equiv \quad \text{wp}(\texttt{"x += 5; y *= 2; z = z \% 4;"}, x + y = 18)$$

Im nächsten Schritt müssten wir ein z verändern. Wir haben aber in userer Bedingung kein z, deshalb kann es wegfallen.

$$\equiv \quad \text{wp}(\texttt{"x += 5; y *= 2;"}, x + y = 18)$$

$y \cdot 2$ einsetzen

$$\equiv \quad \text{wp}(\texttt{"x += 5;"}, x + y \cdot 2 = 18)$$

Auf x wird 5 hinzuaddiert.

$$\equiv \quad \text{wp}(\texttt{""}, x + 5 + y \cdot 2 = 18)$$

Wir haben keinen Programmcode mehr. Wir können wp weglassen.

$$\equiv \quad x + 5 + y \cdot 2 = 18$$

Die 5 nach rechts bringen

$$\equiv \quad x + y \cdot 2 = 13$$

Alle Eingaben die Vorbedingung $P \equiv x + y \cdot 2 = 13$ erfüllen, erfüllen die Nachbedingung Q $x + y = 17$.

Datenfluss-annotierter Kontrollflussgraph

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

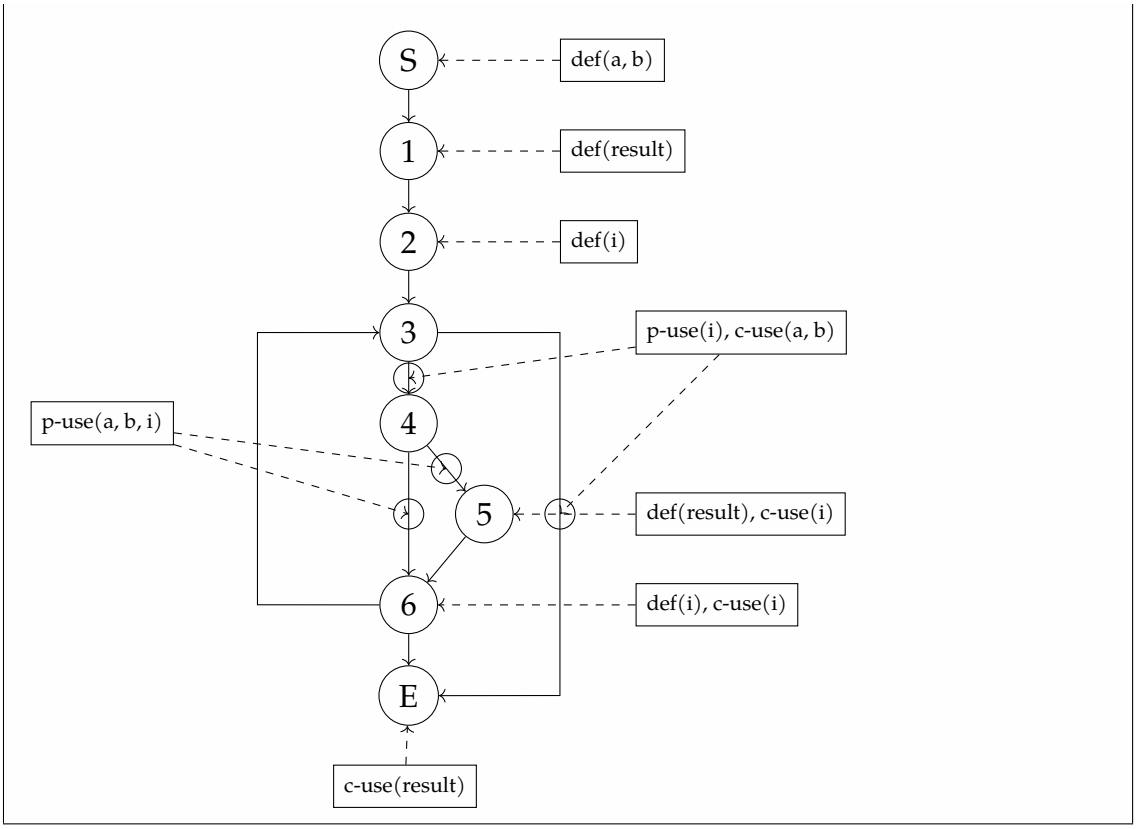
Gegeben sei folgende Methode:

```java
public int ggT(int a, int b) {
  int result = 1;
  for (int i = 1; i <= Math.min(a, b); i++) {
    if ((a % i == 0) & (b % i == 0)) {
      result = i;
    }
  }
  return result;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/sosy/white_box/WhiteBox.java`

(a) Erstellen Sie den zur Methode gehörenden datenflussannotierten Kontrollflussgraphen.

Lösungsvorschlag



(b) Geben Sie die zyklomatische Komplexität M nach McCabe der Methode `ggT` an. (Nur das Ergebnis!)

Lösungsvorschlag

Berechnung durch Anzahl Binärverzweigungen b (p Anzahl der Zusammenhangskomponenten des Kontrollflussgraphen)

$$M = b + p$$

$\rightarrow M = 2 + 1 = 3$

oder durch Anzahl Kanten e und Knoten n

$$M = e - n + 2p$$

$\rightarrow M = 9 - 8 + 2 \cdot 1 = 3$

(c) Geben Sie je einen Repräsentanten aller Pfadklassen im Kontrollflussgraphen an, die zum Erzielen einer vollständigen Schleifen-Inneres-Überdeckung (Boundary-Interior-Coverage) genügen würden.

Lösungsvorschlag

Äußere Pfade

- S 1 2 3 E

Kontrollflussgraph

Grenzpfade

- S 1 2 3 4 5 6 3 E
- S 1 2 3 4 6 3 E

Innere Pfade

- S 1 2 3 4 5 6 3 4 5 6 3 E
- S 1 2 3 4 6 3 4 6 3 E
- S 1 2 3 4 5 6 3 4 6 3 E
- S 1 2 3 4 6 3 4 5 6 3 E

(d) Geben Sie an, welche der Pfade aus der vorherigen Aufgabe nicht überdeckbar ("feasible") sind und begründen Sie dies.

Lösungsvorschlag

Äußere Pfade

S 1 2 3 E ja, z. B. `ggT(-1, -2)`.

Grenzpfade

S 1 2 3 4 5 6 3 E ja, z. B. `ggT(10, 20)`.

S 1 2 3 4 6 3 E ja, z. B. `ggT(1, 2)`.

Innere Pfade

S 1 2 3 4 5 6 3 4 5 6 3 E ja, z. B. `ggT(2, 2)`.

S 1 2 3 4 6 3 4 6 3 E nicht feasible, da geteilt durch eins immer Modulo 0 ergibt, egal welche Zahl a oder b hat. Bei der ersten Schleifenwiederholung wird immer die innere If-Verzweigung genommen.

S 1 2 3 4 5 6 3 4 6 3 E ja, z. B. `ggT(2, 3)`.

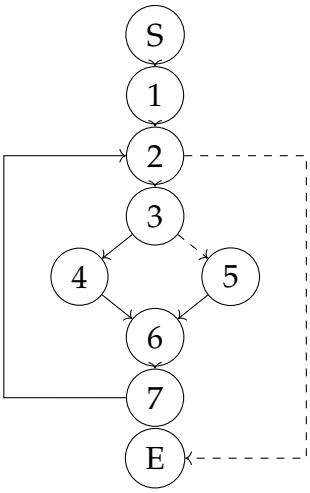
S 1 2 3 4 6 3 4 5 6 3 E nicht feasible, da geteilt durch eins immer Modulo 0 ergibt, egal welche Zahl a oder b hat. Bei der ersten Schleifenwiederholung wird immer die innere If-Verzweigung genommen.

66116 / 2021 / Frühjahr / Thema 1 / Aufgabe 9

Gegeben sei folgende Methode und ihr Kontrollflussgraph:

```java
int log(int a) {
  int x = a;
  int z = 0;
  while (x > 1) {
    if (x % 2 == 0) {
      z++;
      x /= 2;
    } else {
      x--;
    }
  }
  return z;
}
```

Code-Beispiel auf Github ansehen:
`src/main/java/org/bschlangaul/aufgaben/sosy/ab_7/Aufgabe3.java`

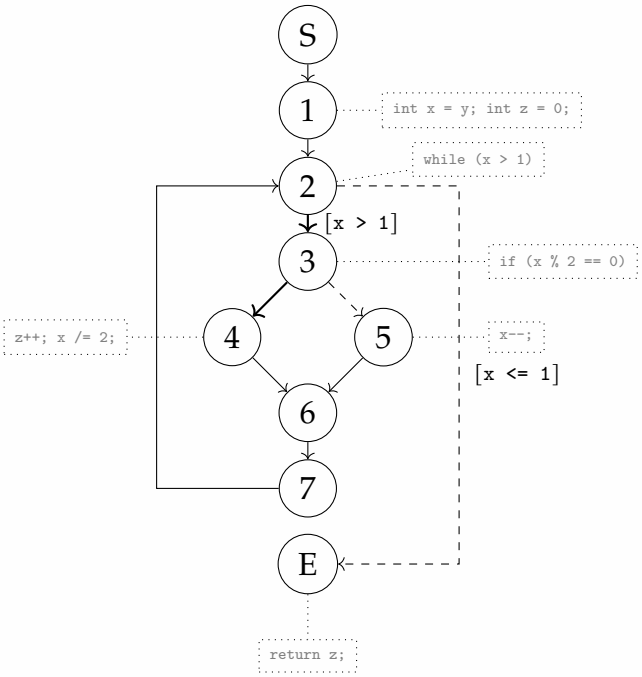


Überdeckbarkeit
C2b Schleife-Inneres-Pfadüberdeckung (Boundary-Interior Path Coverage)

(a) Begründen Sie, warum der Pfad Ⓢ - ① - ② - ③ - ⑤ - ⑥ - ⑦ - ② - Ⓔ infeasible (= nicht überdeckbar) ist, also weshalb es keine Eingabe gibt, unter der dieser Pfad durchlaufen werden kann.

Lösungsvorschlag

Damit dieser Pfad durchlaufen werden könnte, müsste die Eingabe a gleichzeitig 2 und ungerade sein.



(b) Geben Sie eine minimale Menge von Pfaden an, mit der eine vollständigen Schleifen-Inneres-Überdeckung erzielt werden kann, sowie gegebenenfalls zu jedem Pfad eine Eingabe, unter der dieser Pfad durchlaufen werden kann.

Lösungsvorschlag

ohne Schleifenausführung

- Ⓢ - ① - ② - Ⓔ (a=1)

boundary-Tests

- (S) - (1) - (2) - (3) - (4) - (6) - (7) - (2) - (E)
  (a=2)
- (S) - (1) - (2) - (3) - (5) - (6) - (7) - (2) - (E)
  (infeasible)

interior-Tests

- (S) - (1) - (2) - (3) - (4) - (6) - (7) - (2) - (3) - (4) - (6) - (7) - (2) - (E)
  (a=4)
- (S) - (1) - (2) - (3) - (4) - (6) - (7) - (2) - (3) - (5) - (6) - (7) - (2) - (E)
  (a=6)
- (S) - (1) - (2) - (3) - (5) - (6) - (7) - (2) - (3) - (4) - (6) - (7) - (2) - (E)
  (a=3)
- (S) - (1) - (2) - (3) - (5) - (6) - (7) - (2) - (3) - (5) - (6) - (7) - (2) - (E)
  (infeasible)

46115 / 2015 / Herbst / Thema 2 / Aufgabe 4

Gegeben sei die folgende Methode `function`:

```
double function(int n) {
  if (n == 1)
    return 0.5 * n;
  else
    return 1.0 / (n * (n + 1)) + function(n - 1);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2015/herbst/Induktion.java`

Beweisen Sie folgenden Zusammenhang mittels vollständiger Induktion:

$$\forall n \geq 1 \colon \text{function}(n) = f(n) \text{ mit } f(n) := 1 - \frac{1}{n+1}$$

Hinweis: Eventuelle Rechenungenauigkeiten, wie z. B. in Java, bei der Behandlung von Fließkommazahlen (z. B. `double`) sollen beim Beweis nicht berücksichtigt werden - Sie dürfen also annehmen, Fließkommazahlen würden mathematische Genauigkeit aufweisen.

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ———————————

$f(1) := 1 - \frac{1}{1+1} = 1 - \frac{1}{2} = \frac{1}{2}$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ———————————

$f(n) := 1 - \frac{1}{n+1}$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss. —

zu zeigen:

$f(n+1) := 1 - \frac{1}{(n+1)+1} = f(n)$

Vorarbeiten (Java in Mathe umwandeln):

$\text{function}(n) = \frac{1}{n \cdot (n+1)} + f(n-1)$

$$
\begin{aligned}
f(n+1) &= \frac{1}{(n+1)\cdot((n+1)+1)} + f((n+1)-1) && n+1 \text{ eingesetzt} \\
&= \frac{1}{(n+1)\cdot(n+2)} + f(\mathrm{n}) && \text{vereinfacht} \\
&= \frac{1}{(n+1)\cdot(n+2)} + 1 - \frac{1}{n+1} && \text{für } f(n) \text{ Formel eingesetzt} \\
&= 1 + \frac{1}{(n+1)\cdot(n+2)} - \frac{1}{n+1} && \text{1. Bruch an 2. Stelle geschrieben} \\
&= 1 + \frac{1}{(n+1)\cdot(n+2)} - \frac{1\cdot(n+2)}{(n+1)\cdot(n+2)} && \text{2. Bruch mit } (n+2) \text{ erweitert} \\
&= 1 + \frac{1-(n+2)}{(n+1)\cdot(n+2)} && \text{die 2 Brüche subtrahiert} \\
&= 1 + \frac{1-n-2}{(n+1)\cdot(n+2)} && -+2 = -2 \\
&= 1 + \frac{-1-n}{(n+1)\cdot(n+2)} && 1-2 = -1 \\
&= 1 + \frac{-1\cdot(1+n)}{(n+1)\cdot(n+2)} && (n+1) \text{ ausgeklammert} \\
&= 1 + \left(-1 \cdot \frac{(1+n)}{(n+1)\cdot(n+2)}\right) && \text{minus vor den Bruch bringen} \\
&= 1 - \frac{(1+n)}{(n+1)\cdot(n+2)} && \text{plus minus ist minus} \\
&= 1 - \frac{1}{n+2} && (n+1) \text{ gekürzt} \\
&= 1 - \frac{1}{(n+1)+1} && \text{Umformen zur Verdeutlichung}
\end{aligned}
$$

46116 / 2014 / Frühjahr / Thema 2 / Teilaufgabe 1 / Aufgabe 1

Gegeben sei folgende Methode zur Berechnung der Anzahl der notwendigen Züge beim Spiel *„Die Türme von Hanoi“*:

```java
int hanoi(int nr, char from, char to) {
  char free = (char) ('A' + 'B' + 'C' - from - to);
  if (nr > 0) {
    int moves = 1;
    moves += hanoi(nr - 1, from, free);
    System.out.println("Move piece nr. " + nr + " from " + from + " to " + to);
    moves += hanoi(nr - 1, free, to);
    return moves;
  } else {
    return 0;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2014/fruehjahr/Hanoi.java`

(a) Beweisen Sie formal mittels vollständiger Induktion, dass zum Umlegen von k Scheiben (z. B. vom Turm A zum Turm C) insgesamt $2^k - 1$ Schritte notwendig sind, also dass für $k \geq 0$ folgender Zusammenhang gilt:

$$\text{hanoi}(k, \text{'A'}, \text{'C'}) = 2^k - 1$$

Lösungsvorschlag

Zu zeigen:

$$\text{hanoi}(k, \text{'A'}, \text{'C'}) = 2^k - 1$$

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ———————————

$k = 0$

$$\text{hanoi}(0, \text{'A'}, \text{'C'}) = 0$$

$$2^0 - 1 = 1 - 1 = 0$$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ———————————

$$\text{hanoi}(k, \text{'A'}, \text{'C'}) = 2^k - 1$$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss.

Terminierungsfunktion
Kontrollflussgraph

$$\text{hanoi}(k, \text{'A'}, \text{'C'}) = 1 + \text{hanoi}(k-1, \text{'A'}, \text{'B'}) + \text{hanoi}(k-1, \text{'B'}, \text{'C'})$$

$k \to k+1$

$$\begin{aligned}
\text{hanoi}(k+1, \text{'A'}, \text{'C'}) &= 1 + \text{hanoi}((k+1)-1, \text{'A'}, \text{'B'}) + \\
&\quad \text{hanoi}((k+1)-1, \text{'B'}, \text{'C'}) \\
&= 1 + \text{hanoi}(k, \text{'A'}, \text{'B'}) + \\
&\quad \text{hanoi}(k, \text{'B'}, \text{'C'}) && k+1-1 = k \\
&= 1 + 2^k - 1 + 2^k - 1 && \text{Formeln eingesetzt} \\
&= 2^k + 2^k - 1 && 1-1-1 = -1 \\
&= 2 \cdot 2^k - 1 && 2^k + 2^k = 2 \cdot 2^k \\
&= 2^{k+1} - 1 && 2 \cdot 2^k = 2^{k+1}
\end{aligned}$$

(b) Geben Sie eine geeignete Terminierungsfunktion an und begründen Sie kurz Ihre Wahl!

Lösungsvorschlag

Betrachte die Argumentenfolge $k, k-1, k-2, \ldots, 0$. Die Terminierungsfunktion ist offenbar $T(k) = k$. $T(k)$ ist bei jedem Rekursionsschritt auf der Folge der Argumente streng monoton fallend. Bei der impliziten Annahme k ist ganzzahlig und $k \geq 0$ ist $T(k)$ nach unten durch 0 beschränkt.

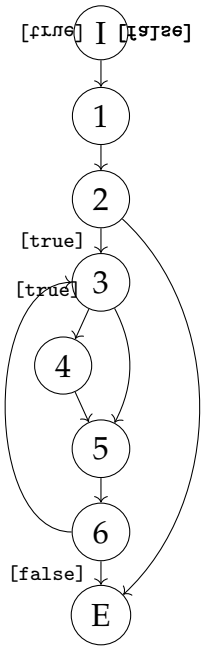
46116 / 2015 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 2

Gegeben sei folgende Methode `isPalindrom` und ihr Kontrollflussgraph:

Abkürzungen: I = Import, E = Export

```java
boolean isPalindrom(String s) {
  boolean yesItIs = true;
  if (s != null && s.length() > 1) {
    do {
      if (s.charAt(0) != s.charAt(s.length() -
↪ 1)) {
        yesItIs = false;
      }
      s = s.substring(1, s.length() - 1);

    } while (yesItIs && s.length() > 1);
  }
  return yesItIs;
}
```

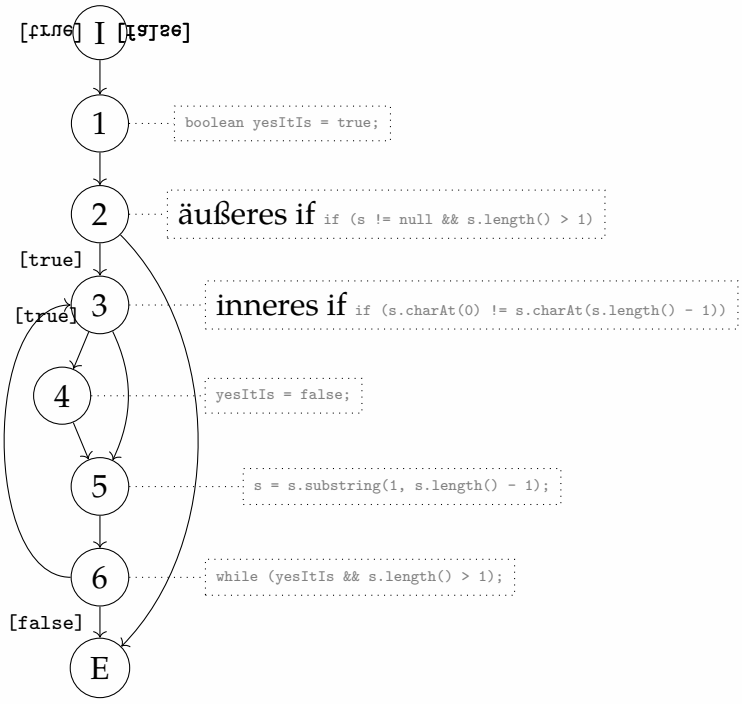
Code-Beispiel auf Github ansehen:
`src/main/java/org/bschlangaul/aufgaben/sosy/pu_5/Aufgabe2.java`



(a) Geben Sie je einen Repräsentanten aller Pfadklassen **im Kontrollflussgraphen** an, die zum Erzielen einer vollständigen ... mit **minimaler** Testfallanzahl und **möglichst kurzen** Pfaden genügen würden.

Lösungsvorschlag

Bemerkung: In der Aufgabenstellung steht *„Geben Sie je einen Repräsentanten aller Pfadklassen **im Kontrollflussgraphen** an, [...] “*. Das bedeutet, dass es hier erstmal egal ist, ob ein Pfad im Code möglich ist oder nicht!



(i) Verzweigungsüberdeckung (Branch-Coverage, C_1)

C1-Test Zweigüberdeckung (Branch Coverage)
C2b Schleife-Inneres-Pfadüberdeckung (Boundary-Interior Path Coverage)

Lösungsvorschlag

Pfad 1 (p1) Ⓘ - ① - ② - Ⓔ
(äußere `if`-Bedingung `false`)

Pfad 2 (p2) Ⓘ - ① - ② - ③ - ⑤ - ⑥ - ③ - ④ - ⑤ - ⑥ - Ⓔ
(äußere `if`-Bedingung `true`, innere `if`-Bedingung `false`, Wiederholung, innere `if`-Bedingung `true`, keine Wiederholung)

(ii) Schleife-Inneres-Überdeckung (Boundary-Interior-Coverage, $C_{\infty,2}$)

Lösungsvorschlag

ohne Ausführung der Wiederholung (äußere Pfade): p1 (siehe oben)
Ⓘ - ① - ② - Ⓔ

Boundary-Test: (alle Pfade, die die Wiederholung betreten, aber nicht wiederholen; innerhalb des Schleifenrumpfes alle Pfade!)

interior-Test: (alle Pfade mit *einer Wiederholung des Schleifenrumpfes*; innerhalb des Schleifenrumpfes wieder alle Pfade!)
innere `if`-Bedingung `true`: ③ - ④ - ⑤ - ⑥
innere `if`-Bedingung `false`: ③ - ⑤ - ⑥

p5 Ⓘ - ① - ② - ③ - ④ - ⑤ - ⑥ - ③ - ④ - ⑤ - ⑥ - Ⓔ
(innere `if`-Bedingung `true`, innere `if`-Bedingung `true`)

p2 (siehe oben) Ⓘ - ① - ② - ③ - ⑤ - ⑥ - ③ - ④ - ⑤ - ⑥ - Ⓔ
(innere `if`-Bedingung `false`, innere `if`-Bedingung `true`)

p6 Ⓘ - ① - ② - ③ - ④ - ⑤ - ⑥ - ③ - ④ - ⑤ - ⑥ - Ⓔ
(innere `if`-Bedingung `true`, innere `if`-Bedingung `false`)

p7 Ⓘ - ① - ② - ③ - ⑤ - ⑥ - ③ - ⑤ - ⑥ - Ⓔ
(innere `if`-Bedingung `false`, innere `if`-Bedingung `false`)

mit **minimaler** Testfallanzahl und **möglichst kurzen** Pfaden genügen würden.

(b) Welche der vorangehend ermittelten Pfade für die $C_{\infty,2}$-Überdeckung sind mittels Testfällen tatsächlich überdeckbar (*„feasible“*)? Falls der Pfad ausführbar ist, geben Sie den zugehörigen Testfall an - andernfalls begründen Sie kurz, weshalb der Pfad nicht überdeckbar ist.

Lösungsvorschlag

p1 `s = "a";`

p2 `s = "abaa";`

p3 `s = "ab";`

p4 `s = "aa";`

p5 nicht überdeckbar, da `yesItIs = false`, wenn innere `if`-Bedingung `true` keine Wiederholung!

p6 nicht überdeckbar, da `yesItIs = false`, wenn innere `if`-Bedingung `true` keine Wiederholung!

```
p7 s = "abba";
```


(c) Bestimmen Sie anhand des Kontrollflussgraphen des obigen Code-Fragments die maximale Anzahl linear unabhängiger Programmpfade, also die zyklomatische Komplexität nach McCabe.

Lösungsvorschlag

$M = b + p = 3 + 1 = 4$

(b: Anzahl Binärverzweigungen, p: Anzahl Zusammenhangskomponenten)

Alternativ

$M = e - n + 2p = 10 - 8 + 2 = 4$

(e: Anzahl Kanten, n: Anzahl Knoten, p: Anzahl Zusammenhangskomponenten)

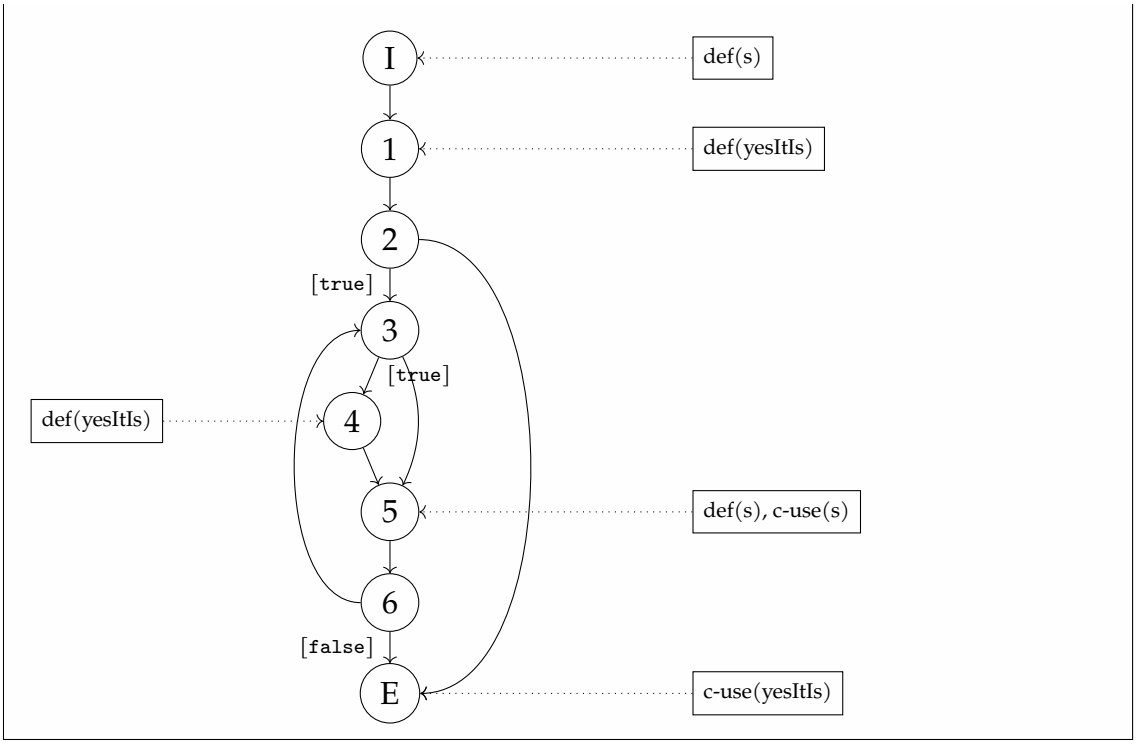
(d) Kann für dieses Modul eine 100%-ige Pfadüberdeckung erzielt werden? Begründen Sie kurz Ihre Antwort.

Lösungsvorschlag

Eine 100%-ige Pfadüberdeckung kann nicht erzielt werden, da es zum einen unüberdeckbare Pfade gibt (vgl. Teilaufgabe b). Zum anderen ist das Testen aller Testfälle nicht möglich, da die Anzahl an Zeichen des übergebenen Wortes nicht begrenzt ist und es somit eine unendliche Anzahl an Testfällen gibt.

(e) Übernehmen Sie den vorgegebenen Kontrollflussgraphen und annotieren Sie ihn mit allen relevanten Datenflussereignissen. Geben Sie jeweils an, ob die Verwendungen berechnend (`c-use`) oder prädikativ (`p-use`) sind.

Lösungsvorschlag



46116 / 2015 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 3

Sei wp(A, Q) die schwächste Vorbedingung (weakest precondition) eines Programmfragments A bei gegebener Nachbedingung Q so, dass A alle Eingaben, die wp(A,Q) erfüllen, auf gültige Ausgaben abbildet, die Q erfüllen.

Bestimmen Sie schrittweise und formal (mittels Floyd-Hoare-Kalkül) jeweils wp(A, Q) für folgende Code-Fragmente A und Nachbedingungen Q und vereinfachen Sie dabei den jeweils ermittelten Ausdruck so weit wie möglich.

Die Variablen x, y und z in folgenden Pseudo-Codes seien ganzzahlig (vom Typ int). Zur Vereinfachung nehmen Sie bitte im Folgenden an, dass die verwendeten Datentypen unbeschränkt sind und daher keine Überläufe auftreten können.

(a) Sequenz:

```
x = -2 * (x + 2 * y);
y += 2 * x + y + z;
z -= x - y - z;
```

$Q :\equiv x = y + z$

Lösungsvorschlag

Code umformulieren:

```
x = -2 * (x + 2 * y);
y = y + 2 * x + y + z;
z = z - (x - y - z);
```

$\mathrm{wp}(\texttt{"x=-2*(x+2*y);y=2*y+2*x+z;z=z-(x-y-z);"}, x = y + z)$

z eingesetzen

$$\equiv \quad \mathrm{wp}(\texttt{"x=-2*(x+2*y);y=2*y+2*x+z;"}, x = y + (z - (x - y - z)))$$

Innere Klammer auflösen

$$\equiv \quad \mathrm{wp}(\texttt{"x=-2*(x+2*y);y=2*y+2*x+z;"}, x = y + (-x + y - 2z))$$

Klammer auflösen

$$\equiv \quad \mathrm{wp}(\texttt{"x=-2*(x+2*y);y=2*y+2*x+z;"}, x = -x + 2y + 2z)$$

$-x$ auf beiden Seiten

$$\equiv \quad \mathrm{wp}(\texttt{"x=-2*(x+2*y);y=2*y+2*x+z;"}, 0 = -2x + 2y + 2z)$$

$\div 2$ auf beiden Seiten

$$\equiv \quad \mathrm{wp}(\texttt{"x=-2*(x+2*y);y=2*y+2*x+z;"}, 0 = -x + y + z)$$

y einsetzen

$$\equiv \quad \mathrm{wp}(\texttt{"x=-2*(x+2*y);"}, 0 = -x + (2y + 2x + z) + z)$$

Term vereinfachen

$$\equiv \quad \mathrm{wp}(\texttt{"x=-2*(x+2*y);"}, 0 = x + 2y + 2z)$$

x einsetzen

$$\equiv \quad \mathrm{wp}(\texttt{""}, 0 = (-2(x + 2y)) + 2y + 2z)$$

wp weglassen

$$\equiv \quad 0 = (-2(x + 2y)) + 2y + 2z$$

ausmultiplizieren

$$\equiv \quad 0 = (-2x - 4y) + 2y + 2z$$

Klammer auflösen, vereinfachen

$$\equiv \quad 0 = -2x - 2y + 2z$$

$\div 2$ auf beiden Seiten

$$\equiv \quad 0 = -x - y + z$$

x nach links holen mit $+x$ auf beiden Seiten

$$\equiv \quad x = -y + z$$

y ganz nach links schreiben

$$\equiv \quad x = z - y$$

$$x = -2 \cdot (x + 2 \cdot y)$$

(b) Verzweigung:

```
if (x < y) {
  x = y + z;
} else if (y > 0) {
  z = y - 1;
} else {
  x -= y -= z;
}
```

$Q :\equiv x > z$

Lösungsvorschlag

1. Fall: $x < y$

2. Fall: $x \geq y \land y > 0$

3. Fall: $x \geq y \land y \leq 0$

Code umformulieren:

```
if (x < y) {
  x = y + z;
} else if (x >= y && y > 0) {
  z = y - 1;
} else {
  y = y - z;
  x = x - y;
}
```

$\mathrm{wp}(\texttt{"if(x<y){x=y+z;}else if(x>=y\&\&y>0){z=y-1;}else{y=y-z;x=x-y;}"}, x > z)$

$\equiv$ (In mehrere kleinere wp-Kalküle aufsplitten)

$$\Big((x < y) \land \mathrm{wp}(\texttt{"x=y+z;"}, x > z)\Big) \lor$$
$$\Big((x \geq y \land y > 0) \land \mathrm{wp}(\texttt{"z=y-1;"}, x > z)\Big) \lor$$
$$\Big((x \geq y \land y \leq 0) \land \mathrm{wp}(\texttt{"y=y-z;x=x-y;"}, x > z)\Big)$$

$\equiv$ (Code einsetzen)

$$\Big((x < y) \wedge \mathrm{wp}(\texttt{""}, y + z > z)\Big) \vee$$
$$\Big((x \geq y \wedge y > 0) \wedge \mathrm{wp}(\texttt{""}, x > y - 1)\Big) \vee$$
$$\Big((x \geq y \wedge y \leq 0) \wedge \mathrm{wp}(\texttt{"y=y-z;"}, x - y > z)\Big)$$

$\equiv$ (wp-Kalkül-Schreibweise weg lassen, Code weiter einsetzen)

$$\Big((x < y) \wedge y + z > z\Big) \vee$$
$$\Big((x \geq y \wedge y > 0) \wedge x > y - 1\Big) \vee$$
$$\Big((x \geq y \wedge y \leq 0) \wedge \mathrm{wp}(\texttt{""}, x - (y - z) > z)\Big)$$

$\equiv$ (Terme vereinfachen, wp-Kalkül-Schreibweise weg lassen)

$$\Big(x < y \wedge y > 0\Big) \vee$$
$$\Big(x \geq y^a \wedge y > 0\Big) \vee$$
$$\Big((x \geq y \wedge y \leq 0) \wedge x - (y - z) > z\Big)$$

$\equiv$ (letzten Term vereinfachen)

$$\Big(x < y \wedge y > 0\Big) \vee$$
$$\Big(x \geq y \wedge y > 0\Big) \vee$$
$$\Big((x \geq y \wedge y \leq 0) \wedge x - y > 0\Big)$$

$\equiv$ (ein $\wedge$ eleminieren)

$$\Big(x < y \land y > 0\Big) \lor$$
$$\Big(x \geq y \land y > 0\Big) \lor$$
$$\Big(y \leq 0 \land x > y\Big)$$

[a] $x > y - 1 \land x \geq y$ ergibt $x \geq y$
[a] $x > y - 1 \land x \geq y$ ergibt $x \geq y$

(c) Mehrfachauswahl:

```
switch (z) {
  case "x":
    y = "x";
  case "y":
    y = --z;
    break;
  default:
    y = 0x39 + "?";
}
```

$Q :\equiv$ `'x'` $= y$

Hinweis zu den ASCII-Codes

- `'x'` $= 120_{(10)}$
- `'y'` $= 121_{(10)}$
- `0x39` $= 57_{(10)}$
- `'?'` $= 63_{(10)}$

Lösungsvorschlag

Mehrfachauswahl in Bedingte Anweisungen umschreiben. Dabei beachten, dass bei fehlendem `break` die Anweisungen im folgenden Fall bzw. ggf. in den folgenden Fällen ausgeführt werden:

```
if (z == "x") {
  y = "x";
  y = z - 1;
} else if (z == "y") {
  y = z - 1;
} else {
  y = 0x39 + "?";
}
```

Da kein `break` im Fall `z == "x"`. `--z` bedeutet, dass die Variable erst um eins verringert und dann zugewiesen wird.

```
if (z == 120) {
  y = 120;
  y = 120 - 1;
} else if (z == 121) {
  y = 121 - 1;
} else {
  y = 57 + 63;
}
```

Vereinfachung / Zusammenfassung:

```
if (z == 120) {
  y = 120;
  y = 119;
} else if (z == 121) {
  y = 120;
} else {
  y = 120;
}
```

$$\text{wp}(\texttt{"if(z==120)\{y=120;y=119;\}else if(z==121)\{y=120;\}else\{y=120;\}"}, 120 = y)$$

$\equiv$ (In mehrere kleinere wp-Kalküle aufsplitten)

$$\begin{aligned}
&\Big((z = 120) \wedge \text{wp}(\texttt{"y=120;y=119;"}, 120 = y)\Big) \vee \\
&\Big(((z \neq 120) \wedge (z = 121)) \wedge \text{wp}(\texttt{"y=120;"}, 120 = y)\Big) \vee \\
&\Big(((z \neq 120) \wedge (z \neq 121)) \wedge \text{wp}(\texttt{"y=120;"}, 120 = y)\Big)
\end{aligned}$$

$\equiv$ (Code einsetzen)

$$\begin{aligned}
&\Big((z = 120) \wedge \text{wp}(\texttt{"y=120;"}, 120 = 119)\Big) \vee \\
&\Big(((z \neq 120) \wedge (z = 121)) \wedge \text{wp}(\texttt{""}, 120 = 120)\Big) \vee \\
&\Big(((z \neq 120) \wedge (z \neq 121)) \wedge \text{wp}(\texttt{""}, 120 = 120)\Big)
\end{aligned}$$

$\equiv$ (vereinfachen)

$$\begin{aligned}
&\text{false} \vee \\
&\Big((z = 121) \wedge \text{true}\Big) \vee \\
&\Big(((z \neq 120) \wedge (z \neq 121)) \wedge \text{true}\Big)
\end{aligned}$$

$\equiv \text{false} \vee (z = 121) \vee ((z \neq 120) \wedge (z \neq 121))$
$\equiv (z = 121) \vee (z \neq 121)$
$\equiv z \neq 121$
Alle Zahlen außer 120 sind möglich bzw. alle Zeichen außer 'x'.

46116 / 2016 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 4

Gegeben sei folgende rekursive Methodendeklaration in der Sprache Java. Es wird als Vorbedingung vorausgesetzt, dass die Methode `cn` nur für Werte $n \geq 0$ aufgerufen wird.

```
int cn(int n) {
  if (n == 0)
    return 1;
  else
    return (4 * (n - 1) + 2) * cn(n - 1) / (n + 1);
}
```

Sie können im Folgenden vereinfachend annehmen, dass es keinen Überlauf in der Berechnung gibt, ðdass der Datentyp `int` für die Berechnung des Ergebnisses stets ausreicht.

(a) Beweisen Sie mittels vollständiger Induktion, dass der Methodenaufruf `cn(n)` für jedes $n \geq 0$ die n-te Catalan-Zahl C_n berechnet, wobei

$$C_n = \frac{(2n)!}{(n+1)! \cdot n!}$$

Exkurs: Fakultät

Für alle natürlichen Zahlen n ist

$$n! = 1 \cdot 2 \cdot 3 \cdots n = \prod_{k=1}^{n} k$$

als das Produkt der natürlichen Zahlen von 1 bis n definiert. Da das leere Produkt stets 1 ist, gilt

$$0! = 1$$

Die Fakultät lässt sich auch rekursiv definieren:

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

Fakultäten für negative oder nicht ganze Zahlen sind nicht definiert. Es gibt aber eine Erweiterung der Fakultät auf solche Argumente [a]

[a] `https://de.wikipedia.org/wiki/Fakultät_(Mathematik)`

Exkurs: Catalan-Zahl

Die Catalan-Zahlen bilden eine Folge natürlicher Zahlen, die in vielen Problemen der Kombinatorik auftritt. Sie sind nach dem belgischen Mathematiker Eugène Charles Catalan benannt.

Die Folge der Catalan-Zahlen $C_0, C_1, C_2, C_3, \ldots$ beginnt mit $1, 1, 2, 5, 14, 42, 132, \ldots$ [a]

[a] `https://de.wikipedia.org/wiki/Catalan-Zahl`

Beim Induktionsschritt können Sie die beiden folgenden Gleichungen verwenden:

(i) $(2(n+1))! = (4n+2) \cdot (n+1) \cdot (2n)!$
(ii) $(a+2)! \cdot (n+1)! = (n+2) \cdot (n+1) \cdot (n+1)! \cdot n!$

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ———————————

$$\begin{aligned} C_0 &= \frac{(2 \cdot 0)!}{(0+1)! \cdot 0!} \\ &= \frac{0!}{1! \cdot 0!} \\ &= \frac{1}{1 \cdot 1} \\ &= \frac{1}{1} \\ &= 1 \end{aligned}$$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ———————————

$$C_n = \frac{(2n)!}{(n+1)! \cdot n!}$$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss.

Vom Code ausgehend

$$
\begin{aligned}
C_{n+1} &= \frac{(4 \cdot (n+1-1)+2) \cdot \mathrm{cn}(n+1-1)}{n+1+1} && \text{Java nach Mathe} \\
&= \frac{(4n+2) \cdot \mathrm{cn}(n)}{n+2} && \text{addiert, subtrahiert} \\
&= \frac{(4n+2) \cdot (2n)!}{(n+2) \cdot (n+1)! \cdot n!} && \text{für cn(n) Formel eingesetzt} \\
&= \frac{(4n+2) \cdot (2n)! \cdot (n+1)}{(n+2) \cdot (n+1)! \cdot n! \cdot (n+1)} && (n+1) \text{ multipliziert} \\
&= \frac{(4n+2) \cdot (n+1) \cdot (2n)!}{(n+2) \cdot (n+1)! \cdot (n+1) \cdot n!} && \text{umsortiert} \\
&= \frac{(2(n+1))!}{(n+2)! \cdot (n+1)!} && \text{Hilfsgleichungen verwendet} \\
&= \frac{(2(n+1))!}{((n+1)+1)! \cdot (n+1)!} && (n+1) \text{ verdeutlicht}
\end{aligned}
$$

Mathematische Herangehensweise

$$
\begin{aligned}
C_{n+1} &= \frac{(2(n+1))!}{((n+1)+1)! \cdot (n+1)!} && n+1 \text{ in } C_n \text{ eingesetzt} \\
&= \frac{(2(n+1))!}{(n+2)! \cdot (n+1)!} && \text{addiert} \\
&= \frac{(4n+2) \cdot (n+1) \cdot (2n)!}{(n+2) \cdot (n+1) \cdot (n+1)! \cdot n!} && \text{Hilfsgleichungen verwendet} \\
&= \frac{(4n+2) \cdot (2n)!}{(n+2) \cdot (n+1)! \cdot n!} && (n+1) \text{ gekürzt} \\
&= \frac{4n+2}{n+2} \cdot C_n && \text{Catalan-Formel ersetzt} \\
&= \frac{4((n+1)-1)+2}{(n+1)+1} \cdot C_{(n+1)-1} && (n+1) \text{ verdeutlicht}
\end{aligned}
$$

(b) Geben Sie eine geeignete Terminierungsfunktion an und begründen Sie, warum der Methodenaufruf `cn(n)` für jedes $n \geq 0$ terminiert.

Bubblesort
Black-Box-Testing

Lösungsvorschlag

$T(n) = n$. Diese Funktion verringert sich bei jedem Rekursionsschritt um eins. Sie ist monoton fallend und für $T(0) = 0$ definiert. Damit ist sie eine Terminierungsfunktion für `cn(n)` .

46116 / 2017 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 4

Ein gängiger Ansatz zur Messung der Qualität von Software ist das automatisierte Testen von Programmen. Im Folgenden werden praktische Testmethoden anhand des nachstehend angegebenen Sortieralgorithmus diskutiert.

Algorithmus 1 Bubble Sort

```
public class BubbleSort {
  void bubblesort(int[] array, int len) {
    for (int i = 0; i < len - 1; i++) {   // 1
      for (int j = 0; j < len - 1; j++) { // 2
        if (array[j] > array[j + 1]) {    // 3
          int temp = array[j];            // 4
          array[j] = array[j + 1];        // 5
          array[j + 1] = temp;            // 6
        }
      }
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46116/jahr_2017/herbst/BubbleSort.java`

(a) Nennen Sie eine Art des Black-Box-Testens und beschreiben Sie deren Durchführung anhand des vorgegebenen Algorithmus.

Lösungsvorschlag

Beim Black-Box-Testen sind die Testfälle von Daten getrieben (Data-Driven) und beziehen sich auf die Anforderungen und das spezifizierte Verhalten.)

$\Rightarrow$ Aufruf der Methoden mit verschiedenen Eingangsparametern und Vergleich der erhaltenen Ergebnisse mit den erwarteten Ergebnissen.

Das Ziel ist dabei eine möglichst hohe Anforderungsüberdeckung, wobei man eine minimale Anzahl von Testfällen durch Äquivalenzklassenzerlegung (1) und Grenzwertanalyse (2) erhält.

zu (1): Man identifiziert Bereiche von Eingabewerten, die jeweils diesselben Ergebnisse liefern. Dies sind die sog. Äquivalenzklassen. Aus diesen wählt man nun je einen Repräsentanten und nutzen diesen für den Testfall.

zu (2): Bei der Grenzwertanalyse identifiziert man die Grenzbereiche der Eingabedaten und wählt Daten aus dem nahen Umfeld dieser für seine Testfälle.

Kontrollflussgraph
C2a Vollständige Pfadüberdeckung (Full Path Coverage)
Vollständige Induktion

Angewendet auf den gegebenen Bubblesort-Algorithmus würde die Grenzwertanalyse bedeuten, dass man ein bereits aufsteigend sortiertes Array und ein absteigend sortiertes Array übergibt.

(b) Zeichnen Sie ein mit Zeilennummem beschriftetes Kontrollflussdiagramm für den oben angegebenen Sortieralgorithmus.

Lösungsvorschlag

Zur Erinnerung: Eine im Code enthaltene Wiederholung mit for muss wie folgt im Kontrollflussgraphen „zerlegt“ werden:

(c) Erklären Sie, ob eine vollständige Pfadüberdeckung für die gegebene Funktion möglich und sinnvoll ist.

Lösungsvorschlag

Eine vollständige Pfadüberdeckung (C_1-Test) kann nicht erreicht werden, da die Bedingung der inneren Wiederholung immer wahr ist, wenn die Bedingung der äußeren Wiederholung wahr ist. D. h., der Pfad S-1-1-2-2-1“ kann nie gegangen werden. Dies wäre aber auch nicht sinnvoll, weil jeder Eintrag mit jedem anderen verglichen werden soll und im Fall true $\rightarrow$ false ein Durchgang ausgelassen.

66112 / 2003 / Herbst / Thema 2 / Aufgabe 5

Zeigen Sie mit Hilfe vollständiger Induktion, dass das folgende Programm bzgl. der Vorbedingung $x > 0$ und der Nachbedingung drei_hoch $x = 3^x$ partiell korrekt ist!

```
(define (drei_hoch x)
  (cond ((= x 0) 1)
    (else (* 3 (drei_hoch (- x 1))))
  )
)
```

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ——————

drei_hoch $1 = 3 \cdot (\text{drei\_hoch}\, 0) = 3 \cdot 1 = 3$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ——————

für alle $x < x0$ gilt drei_hoch $x = 3x$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss. ——

x->x+1

$$\begin{aligned}\text{drei\_hoch}\,(x+1) &= 3 \cdot \text{drei\_hoch}\,(-(x+1)1)) \\ &= 3 \cdot (\text{drei\_hoch}\,x) \\ &= 3 \cdot 3^x \\ &= 3^{x+1}\end{aligned}$$

66115 / 2014 / Frühjahr / Thema 1 / Aufgabe 1

(a) Gegeben sei die Methode `BigInteger lfBig(int n)` zur Berechnung der eingeschränkten Linksfakultät:

$$!n := \begin{cases} n!(n-1) - (n-1)!(n-2) & \text{falls } 1 < n < 32767 \\ 1 & \text{falls } n = 1 \\ 0 & \text{sonst} \end{cases}$$

```
import java.math.BigInteger;
import static java.math.BigInteger.ZERO;
import static java.math.BigInteger.ONE;

public class LeftFactorial {

  BigInteger sub(BigInteger a, BigInteger b) {
    return a.subtract(b);
  }

  BigInteger mul(BigInteger a, BigInteger b) {
    return a.multiply(b);
  }

  BigInteger mul(int a, BigInteger b) {
    return mul(BigInteger.valueOf(a), b);
  }

  // returns the left factorial !n
  BigInteger lfBig(int n) {
    if (n <= 0 || n >= Short.MAX_VALUE) {
      return ZERO;
    } else if (n == 1) {
      return ONE;
    } else {
      return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
    }
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

Implementieren Sie unter Verwendung des Konzeptes der *dynamischen Programmierung* die Methode `BigInteger dp(int n)`, die jede $!n$ auch bei mehrfachem

Aufrufen mit dem gleichen Parameter höchstens einmal rekursiv berechnet. Sie dürfen der Klasse `LeftFactorial` genau ein Attribut beliebigen Datentyps hinzufügen und die in `lfBig(int)` verwendeten Methoden und Konstanten ebenfalls nutzen.

Lösungsvorschlag

Wir führen ein Attribut mit dem Namen `store` ein und erzeugen ein Feld vom Typ `BigInteger` mit der Länge $n+1$. Die Länge des Feld $n+1$ hat den Vorteil, dass nicht ständig $n-1$ verwendet werden muss, um den gewünschten Wert zu erhalten.

In der untenstehenden Implementation gibt es zwei Methoden mit dem Namen `dp`. Die untenstehende Methode ist nur eine Hüllmethode, mit der nach außen hin die Berechnung gestartet und das `store`-Feld neu gesetzt wird. So ist es möglich `dp()` mehrmals hintereinander mit verschiedenen Werten aufzurufen (siehe `main()`-Methode).

```
BigInteger[] store;

BigInteger dp(int n, BigInteger[] store) {
  if (n > 1 && store[n] != null) {
    return store[n];
  }
  if (n <= 0 || n >= Short.MAX_VALUE) {
    return ZERO;
  } else if (n == 1) {
    return ONE;
  } else {
    BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1, dp(n - 2,
↪ store)));
    store[n] = result;
    return result;
  }
}

BigInteger dp(int n) {
  store = new BigInteger[n + 1];
  return dp(n, store);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

(b) Betrachten Sie nun die Methode `lfLong(int)` zur Berechnung der vorangehend definierten Linksfakultät ohne obere Schranke. Nehmen Sie im Folgenden an, dass der Datentyp `long` unbeschränkt ist und daher kein Überlauf auftritt.

```
long lfLong(int n) {
  if (n <= 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
```

```
    return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

Beweisen Sie *formal* mittels *vollständiger Induktion*:

$$\forall n \geq 0 : \text{lfLong}(n) \equiv \sum_{k=0}^{n-1} k!$$

Lösungsvorschlag

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ———————————

$$n = 1 \Rightarrow \texttt{lfLong(1)} = 1 = \sum_{k=0}^{n-1} k! = 0! = 1$$

$$\begin{aligned} n = 2 \Rightarrow & \text{lfLong}(2) \\ & = (n+1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! \\ & = \texttt{2 * lfLong(1) - 1 * lfLong(0)} \\ & = 2 \\ & = \sum_{k=0}^{1} k! \\ & = 1! + 0! \\ & = 1 + 1 \\ & = 2 \end{aligned}$$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ———————————

$$\texttt{lfLong(n)} = \sum_{k=0}^{n-1} k!$$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k+1)$ wahr sein muss.

$$
\begin{aligned}
&A(n+1) \\
&\quad = \texttt{lfLong(n + 1)} \\
&\quad = \texttt{(n + 1) * lfLong(n) - n * lfLong(n - 1)} \\
&\quad = (n+1)\sum_{k=0}^{n-1} k! - n\sum_{k=0}^{(n-1)-1} k! && \text{Formel eingesetzt} \\
&\quad = (n+1)\sum_{k=0}^{n-1} k! - n\sum_{k=0}^{n-2} k! && \text{subtrahiert} \\
&\quad = n\sum_{k=0}^{n-1} k! + \sum_{k=0}^{n-1} k! - n\sum_{k=0}^{n-2} k! && \text{ausmultipliziert mit } (n+1) \\
&\quad = \sum_{k=0}^{n-1} k! + n\sum_{k=0}^{n-1} k! - n\sum_{k=0}^{n-2} k! && \text{Reihenfolge der Terme geändert} \\
&\quad = \sum_{k=0}^{n-1} k! + n\left((n-1)! + \sum_{k=0}^{n-2} k!\right) - n\sum_{k=0}^{n-2} k! && (n-1)! \text{ aus Summenzeichen entfernt} \\
&\quad = \sum_{k=0}^{n-1} k! + n\left((n-1)! + \sum_{k=0}^{n-2} k! - \sum_{k=0}^{n-2} k!\right) && \text{Distributivgesetz } ac - bc = (a-b)c \\
&\quad = \sum_{k=0}^{n-1} k! + n(n-1)! && +\Sigma - \Sigma = 0 \\
&\quad = \sum_{k=0}^{n-1} k! + n! && \text{Fakultät erhöht} \\
&\quad = \sum_{k=0}^{n} k! && \text{Element zum Summenzeichen hinzugefügt} \\
&\quad = \sum_{k=0}^{(n+1)-1} k! && \text{mit } (n+1) \text{ an der Stelle von } n
\end{aligned}
$$

66115 / 2017 / Frühjahr / Thema 1 / Aufgabe 4

Sie dürfen im Folgenden davon ausgehen, dass keinerlei Under- oder Overflows auftreten.

Gegeben sei folgende rekursive Methode für $n \geq 0$:

```
long sumOfSquares (long n) {
  if (n == 0)
    return 0;
```

```
    else
      return n * n + sumOfSquares(n - 1);
}
```

(a) Beweisen Sie formal mittels vollständiger Induktion:

$$\forall n \in \mathbb{N} : \texttt{sumOfSquares(n)} = \frac{n(n+1)(2n+1)}{6}$$

Lösungsvorschlag

Sei $f(n) : \frac{n(n+1)(2n+1)}{6}$

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist. ———————————

Für $n = 0$ gilt:

$\texttt{sumOfSquares(0)} \overset{\texttt{if}}{=} 0 = f(0)$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$. ———————————

Für ein festes $n \in \mathbb{N}$ gelte:

$\texttt{sumOfSquares(n)} = f(n)$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss.

$n \to n + 1$

Kontrollflussorientieres Testen

$$
\begin{aligned}
f(n+1) &= \texttt{sumOfSquares(n+1)} && \text{Java-Methode eingesetzt}\\
&\overset{\texttt{else}}{=} \texttt{(n+1)*(n+1) + sumOfSquares(n)} && \text{Java-Code der else-Verzweigung verwendet}\\
&\overset{\texttt{I.H.}}{=} (n+1)(n+1)+f(n) && \text{mathematisch notiert}\\
&= (n+1)(n+1)+\frac{n(n+1)(2n+1)}{6} && \text{Formel eingesetzt}\\
&= (n+1)^2+\frac{n(n+1)(2n+1)}{6} && \text{potenziert}\\
&= \frac{6(n+1)^2}{6}+\frac{n(n+1)(2n+1)}{6} && (n+1)^2 \text{ in Bruch umgewandelt}\\
&= \frac{6(n+1)^2+n(n+1)(2n+1)}{6} && \text{Addition gleichnamiger Brüche}\\
&= \frac{(n+1)6(n+1)+(n+1)n(2n+1)}{6} && n+1 \text{ ausklammern vorbereitet}\\
&= \frac{(n+1)(6(n+1)+n(2n+1))}{6} && n+1 \text{ ausgeklammert}\\
&= \frac{(n+1)(6n+6+2n^2+n))}{6} && \text{Klammern ausmultipliziert / aufgelöst}\\
&= \frac{(n+1)(2n^2+7n+6)}{6} && \text{umsortiert, addiert } 6n+n=7n\\
&= \frac{(n+1)(2n^2+3n+4n+6)}{6} && \text{Ausklammern vorbereitet}\\
&= \frac{(n+1)(n+2)(2n+3)}{6} && (n+2) \text{ ausgeklammert}\\
&= \frac{(n+1)((n+1)+1)(2(n+1)+1))}{6} && (n+1) \text{ verdeutlicht}
\end{aligned}
$$

[a]

[a] https://mathcs.org/analysis/reals/infinity/answers/sm_sq_cb.html

(b) Beweisen Sie die Terminierung von `sumOfSquares(n)` für alle $n \geq 0$.

Lösungsvorschlag

Sei $T(n) = n$. Die Funktion $T(n)$ ist offenbar ganzzahlig. In jedem Rekursionsschritt wird n um eins verringert, somit ist $T(n)$ streng monoton fallend. Durch die Abbruchbedingung `n==0` ist $T(n)$ insbesondere nach unten beschränkt. Somit ist T eine gültige Terminierungsfunktion.

66116 / 2014 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 3

Im Folgenden ist ein Algorithmus angegeben, der für eine positive Zahl `until` die Summe aller Zahlen bildet, die kleiner als `until` und Vielfache von 4 oder 6 sind. Für

nicht positive Zahlen soll 0 zurückgegeben werden. Der Algorithmus soll also folgender Spezifikation genügen:

Kontrollflussgraph
C0-Test
Anweisungsüberdeckung (Statement Coverage)

$\texttt{until} > 0 \Rightarrow \texttt{specialSums(until)} = \sum\{y \mid 0 < y < \texttt{until} \wedge (y\%4 = 0 \vee y\%6 = 0)\}$
$\texttt{until} \leq 0 \Rightarrow \texttt{specialSums(until)} = 0$

wobei % den Modulo-Operator bezeichnet.

```java
public static long specialSums(int until) {
  long sum = 0; // 0
  if (until > 0) { // 1
    for (int i = 1; i <= until; i++) { // 2 // 5
      if (i % 4 == 0 || i % 6 == 0) { // 3
        sum += i; // 4
      }
    }
  }
  return sum; // 6
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2014/herbst/SpecialSum.java`

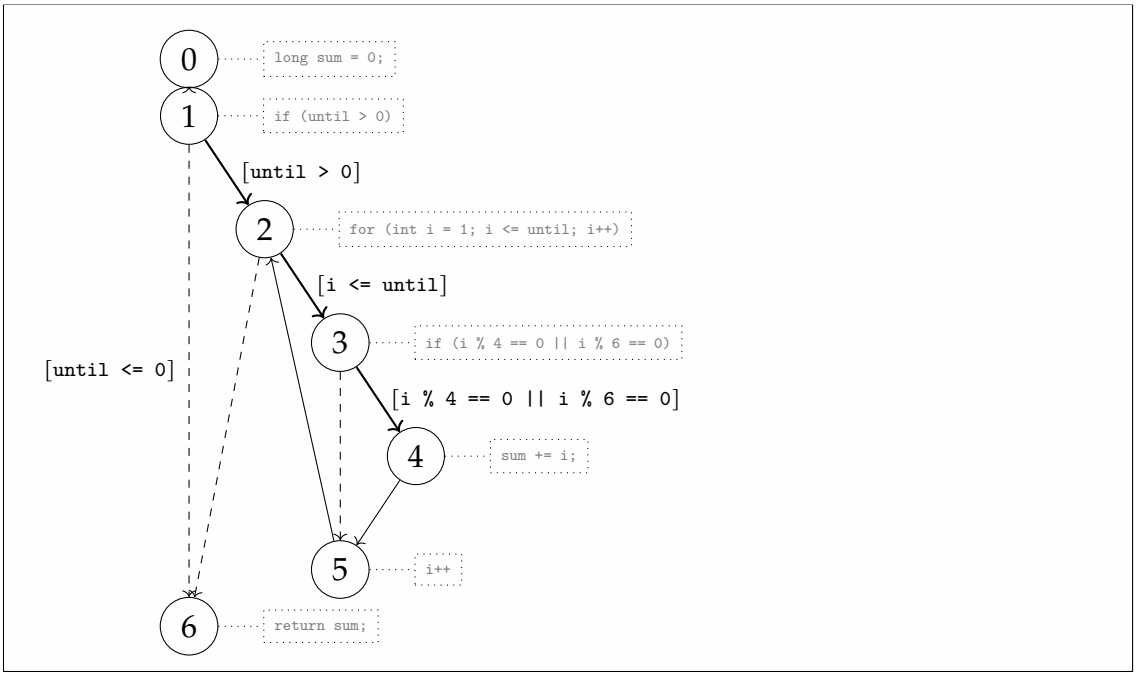
Beachten Sie, dass der Algorithmus nicht der Spezifikation genügt. Der Fehler liegt in der Bedingung der for-Schleife. Der Fehler kann jedoch einfach korrigiert werden indem die Bedingung

$$i \leq \texttt{until} \text{ in } i < \texttt{until}$$

geändert würde.

(a) Zeichnen Sie das zum Programm gehörige Ablaufdiagramm.

Lösungsvorschlag



(b) Schreiben Sie einen Testfall, der das Kriterium „100% Anweisungsüberdeckung“ erfüllt, aber den Fehler trotzdem nicht aufdeckt.

Lösungsvorschlag

C1-Test Zweigüberdeckung (Branch Coverage)

Der Fehler fällt nur dann auf, wenn `until` durch 4 oder 6 ohne Rest teilbar ist. `until = 0%4` oder `until = 0%6`. Wähle daher den Testfall $\{(1,0)\}$. Alternativ kann für die Eingabe auch 2, 3, 5, 7, 9, 10, 11, 13, ... gewählt werden.

(c) Schreiben Sie einen Testfall, der das Kriterium „100% Zweigüberdeckung“ erfüllt, aber den Fehler trotzdem nicht aufdeckt.

Lösungsvorschlag

Betrachte den Testfall $\{(0,0),(5,4)\}$.

erste if-Bedingung Das erste Tupel mit until $= 0$ stellt sicher, dass die erste `if`-Bedingung `false` wird.

Bedingung der for-Schleife Für die zweite Eingabe until $= 5$ werden für i die Werte $1,2,3,4,5,6$ angenommen. Wobei für $i = 6$ die Bedingung der for-Schleife `false` ist.

Innere if-Bedingung Für $i = 1,2,3,5$ wird die innere if-Bedingung jeweils `false`, für $i = 4$ wird sie `true`.

(d) Schreiben Sie einen Testfall, der den Fehler aufdeckt. Berechnen Sie Anweisungsüberdeckung und Zweigüberdeckung ihres Testfalls.

Lösungsvorschlag

Anweisungsüberdeckung Wähle $\{(4,0)\}$. Durch die fehlerhafte Bedingung in der for-Schleife wird der Wert $i = 4$ akzeptiert. Da alle Anweisungen ausgeführt werden, wird eine Anweisungsüberdeckung mit 100% erreicht.

Verzweigungsüberdeckung Da die erste Verzweigung nur zur Hälfte überdeckt wird und die anderen beiden vollständig, gilt für die Verzweigungsüberdeckung:
$\frac{1+2+2}{2+2+2} = \frac{5}{6}$

(e) Es ist nicht immer möglich vollständige Pfadüberdeckung zu erreichen. Geben Sie einen gültigen Pfad des Programmes an, der nicht erreichbar ist. Ein Testfall kann als Menge von Paaren dargestellt werden, wobei jedes Paar (I,O) die Eingabe I und die zu dieser erwartete Ausgabe O darstellt.

Lösungsvorschlag

Ein gültiger Pfad im Kontrollflussgraphen wäre ⓪ - ① - ② - ③ - ④ - ⑤ - ② - ⑥. Der Übergang von ③ auf ④ ist hier aber nicht möglich, da beim ersten Durchlaufen der for-Schleife (②) i immer 1 ist und 1 weder durch 4 noch durch 6 teilbar ist. Somit kann die Bedingung des inneren ifs (③) beim ersten Durchlauf nie *wahr* sein, womit immer der Übergang ③ - ⑤ zu Beginn genommen werden muss. Alle Pfade, die zu Beginn ③ - ④ enthalten, sind somit nicht

überdeckbar.

66116 / 2015 / Herbst / Thema 2 / Teilaufgabe 2 / Aufgabe 3

Gegeben Sei folgendes Programm:

```
long doubleFac (long n) {
  /* P */ long df = 1;
  for (long x = n; x > 1; x -= 2) {
    df *= x;
  } /* Q */
  return df;
}
```

sowie die Vorbedingung $P \equiv n \geq 0$ und Nachbedingung $Q \equiv (\text{df} = n!!)$ wobei gilt

$$n!! := \begin{cases} 2^k \cdot k! & n \text{ gerade}, k := \frac{n}{2} \\ \frac{(2k)!}{2^k \cdot k!} & n \text{ ungerade}, k := \frac{n+1}{2} \end{cases}$$

Exkurs: Fakultät
Die Fakultät ist eine Funktion, die einer natürlichen Zahl das Produkt aller natürlichen Zahlen (ohne Null) kleiner und gleich dieser Zahl zuordnet. Für alle natürlichen Zahlen n ist

$$n! = 1 \cdot 2 \cdot 3 \cdots n = \prod_{k=1}^{n} k$$

Exkurs: Doppelfakultät
Die seltener verwendete „Doppelfakultät“ oder „doppelte Fakultät“ ist für gerade n das Produkt aller geraden Zahlen kleiner gleich n. Für ungerade n ist es das Produkt aller ungeraden Zahlen kleiner gleich n. Sie ist definiert als:

$$n!! = \begin{cases} n \cdot (n-2) \cdot (n-4) \cdots 2 & \text{für } n \text{ gerade und } n > 0, \\ n \cdot (n-2) \cdot (n-4) \cdots 1 & \text{für } n \text{ ungerade und } n > 0, \\ 1 & \text{für } n \in \{-1, 0\} \end{cases}$$

Häufig werden anstelle der Doppelfakultät Ausdrücke mit der gewöhnlichen Fakultät verwendet. Es gilt $(2k)!! = 2^k k!$ und $(2k-1)!! = \frac{(2k)!}{2^k k!}$

Zur Vereinfachung nehmen Sie im Folgenden an, dass die verwendeten Datentypen unbeschränkt sind und daher keine Überläufe auftreten können.

(a) Welche der folgenden Bedingungen ist eine zum Beweisen der Korrektheit der Methode mittels wp-Kalkül (Floyd-Hoare-Kalkül) sinnvolle Schleifeninvariante?

(i) $\text{df} = n!! - x!! \wedge x \geq 1$

(ii) $\text{df} = (n-x)!! \wedge x \geq 1$

(iii) $\mathrm{df} \cdot x!! = n!! \wedge x \geq 0$

(iv) $(\mathrm{df} + x)!! = n!! \wedge x \geq 0$

Lösungsvorschlag

Zunächst wird der Code in einen äquivalenten Code mit while-Schleife umgewandelt:

```
long doubleFac (long n) {
  /* P */ long df = 1;
  long x = n ;
  while (x > 1) {
    df = df * x;
    x = x - 2;
  } /* Q */
  return df;
}
```

(i) $\mathrm{df} = n!! - x!! \wedge x \geq 1$

(ii) $\mathrm{df} = (n - x)!! \wedge x \geq 1$

Die ersten beiden Bedingungen sind unmöglich, da z. B. für $n = 2$ nach der Schleife $x = 0$ gilt und daher $x \geq 1$ verletzt wäre.

(iii) $\mathrm{df} \cdot x!! = n!! \wedge x \geq 0$

Nach dem Ausschlussprinzip ist es daher die dritte Bedingung: $I \equiv (\mathrm{df} + x)!! = n!! \wedge x \geq 0$.

(iv) $(\mathrm{df} + x)!! = n!! \wedge x \geq 0$

Die letzte kann es auch nicht sein, da vor der Schleife $\mathrm{df} = 1$ und $x = n$ gilt, ð$(\mathrm{df} + x)!! = (1 + n)!!$. Jedoch ist offenbar $(1 + n)!! \neq n!!$.

$\Rightarrow$ Die Schleifeninvariante lautet: $\mathrm{df} \cdot x!! = n!! \wedge x \geq 0$

(b) Zeigen Sie formal mittels wp-Kalkül, dass die von Ihnen gewählte Bedingung unmittelbar vor Beginn der Schleife gilt, wenn zu Beginn der Methode die Anfangsbedingung P gilt.

Lösungsvorschlag

Zu zeigen $P \Rightarrow \mathrm{wp}(\texttt{"Code vor der Schleife"}, I)$

$$\begin{aligned}
\text{wp}(\texttt{"Code vor der Schleife"}, I) &\equiv \text{wp}(\texttt{"df = 1; x = n;"}, (\text{df} \cdot x)!! = n!! \land x \geq 0) \\
&\equiv \text{wp}(\texttt{"df = 1;"}, (\text{df} \cdot n)!! = n!! \land n \geq 0) \\
&\equiv \text{wp}(\texttt{""}, (1 \cdot n)!! = n!! \land n \geq 0) \\
&\equiv n!! = n!! \land n \geq 0 \\
&\equiv n \geq 0 \\
&\equiv P
\end{aligned}$$

Insbesondere folgt damit die Behauptung.

(c) Zeigen Sie formal mittels wp-Kalkül, dass die von Ihnen gewählte Bedingung tatsächlich eine Invariante der Schleife ist.

Lösungsvorschlag

zu zeigen: $I \land \text{Schleifenbedingung} \Rightarrow \text{wp}(\texttt{"Code in der Schleife"}, I)$

Bevor wir dies beweisen, zeigen wir erst $x \cdot (x-2)!! = x!!$.

- Fall x ist gerade ($n!! = 2^k \cdot k!$ für $k := \frac{n}{2}$):
  $x \cdot (x-2)!! = x \cdot 2^{\frac{x-2}{2}} \cdot (\frac{x-2}{2})! = x \cdot \frac{1}{2} \cdot 2^{\frac{x}{2}} \cdot (\frac{x}{2}-1)! = 2^{\frac{x}{2}} \cdot (\frac{x}{2})! = x!!$
  Nebenrechnung (Division mit gleicher Basis: $x^{a-b} = \frac{x^a}{x^b}$):
  $2^{\frac{x-2}{2}} = 2^{(\frac{x}{2}-\frac{2}{2})} = \frac{2^{\frac{x}{2}}}{2^{\frac{2}{2}}} = \frac{2^{\frac{x}{2}}}{2^1} = \frac{2^{\frac{x}{2}}}{2} = \frac{1}{2} \cdot 2^{\frac{x}{2}}$
  Nebenrechnung ($n! = (n-1)! \cdot n$):
  $x \cdot \frac{1}{2} \cdot (\frac{x}{2}-1)! = \frac{x}{2} \cdot (\frac{x}{2}-1)! = \frac{x}{2}!$
- Fall x ist ungerade:

Dies benutzen wir nun, um den eigentlichen Beweis zu führen:

$$\begin{aligned}
\text{wp}(\texttt{"Code vor der Schleife"}, I) &\equiv \text{wp}(\texttt{"df = df * x; x = x - 2;"}, (\text{df} \cdot x)!! = n!! \land x \geq 0) \\
&\equiv \text{wp}(\texttt{"df = df * x;"}, (\text{df} \cdot (x-2)))!! = n!! \land x - 2 \geq 0) \\
&\equiv \text{wp}(\texttt{""}, (\text{df} \cdot x \cdot (x-2)))!! = n!! \land x - 2 \geq 0) \\
&\equiv (\text{df} \cdot x)!! = n!! \land x \geq 2 \\
&\equiv (\text{df} \cdot x)!! = n!! \land x > 1 \\
&\equiv I \land x > 1 \\
&\equiv I \land \text{Schleifenbedingung}
\end{aligned}$$

(d) Zeigen Sie formal mittels wp-Kalkül, dass am Ende der Methode die Nachbedingung Q erfüllt wird.

Lösungsvorschlag

Terminierungsfunktion
Datenflussorientiertes Testen
Bubblesort
Datenfluss-annotierter Kontrollflussgraph

z.z. $I \wedge \neg$Schleifenbedingung $\Rightarrow \text{wp}(\texttt{"Code nach der Schleife"}, Q)$

Wir vereinfachen den Ausdruck $I \wedge \neg$Schleifenbedingung:

$I \wedge \neg\text{Schleifenbedingung} \equiv I \wedge (x \leq 1) \equiv I \wedge ((x = 0) \vee (x = 1)) \equiv (I \wedge (x = 0)) \vee (I \wedge (x = 1)) \equiv (df \cdot 1 = n!!) \vee (df \cdot 1 = n!!) \equiv df = n!!$

Damit gilt:

$\text{wp}(\texttt{"Code nach der Schleife"}, Q) \equiv \text{wp}(\texttt{""}, df = n!!) \equiv df = n!! \equiv I \wedge \neg$Schleifenbedingung

(e) Beweisen Sie, dass die Methode immer terminiert. Geben Sie dazu eine Terminierungsfunktion an und begründen Sie kurz ihre Wahl.

Lösungsvorschlag

Sei $T(x) := x$. T ist offenbar ganzzahlig. Da x in jedem Schleifendurchlauf um 2 verringert wird, ist T streng monoton fallend. Aus der Schleifeninvariante folgt $x \geq 0$ und daher ist x auch nach unten beschränkt. Damit folgt $I \Rightarrow T \geq 0$ und T ist eine gültige Terminierungsfunktion.

66116 / 2016 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 3

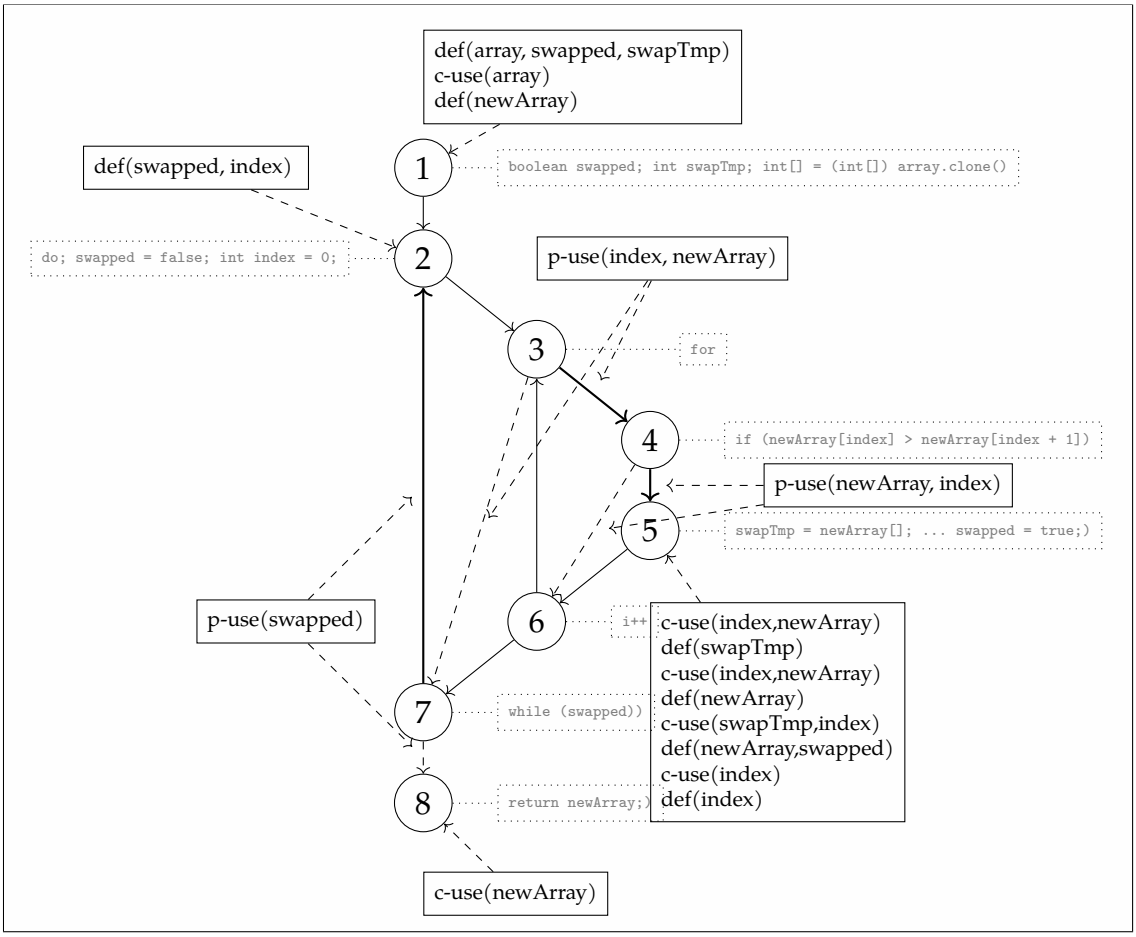
Gegeben Sei folgende Java-Methode `sort` zum Sortieren eines Feldes ganzer Zahlen:

```
  public static int[] sort(int[] array) {
    boolean swapped;
    int swapTmp;
    int[] newArray = (int[]) array.clone();
    do {
      swapped = false;
      for (int index = 0; index < newArray.length - 1; index++) {
        if (newArray[index] > newArray[index + 1]) {
          swapTmp = newArray[index];
          newArray[index] = newArray[index + 1];
          newArray[index + 1] = swapTmp;
          swapped = true;
        }
      }
    } while (swapped);
    return newArray;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66116/jahr_2016/herbst/BubbleSort.java`

(a) Konstruieren Sie den Kontrollflussgraphen des obigen Code-Fragments und annotieren Sie an den Knoten und Kanten die zugehörigen Datenflussinformationen (Definitionen bzw. berechnende oder prädikative Verwendung von Variablen).

Lösungsvorschlag



(b) Nennen Sie die maximale Anzahl linear unabhängiger Programmpfade, also die zyklomatische Komplexität nach McCabe.

Lösungsvorschlag

Der Graph hat 8 Knoten und 10 Kanten. Daher ist die zyklomatische Komplexität nach McCabe gegeben durch 10 - 8 + 2 = 4.

(c) Geben Sie einen möglichst kleinen Testdatensatz an, der eine 100%-ige Verzweigungsüberdeckung dieses Moduls erzielt.

Lösungsvorschlag

Die Eingabe muss mindestens ein Feld der Länge 3 sein. Ansonsten wäre das Feld schon sortiert bzw. bräuchte nur eine Vertauschung und die innere if-Bedingung wäre nicht zu 100% überdeckt. Daher wählt man beispielsweise array = [1,3,2].

(d) Beschreiben Sie kurz, welche Eigenschaften eine Testfallmenge allgemein haben muss, damit das datenflussorientierte Überdeckungskriterium „all-uses“ erfüllt.

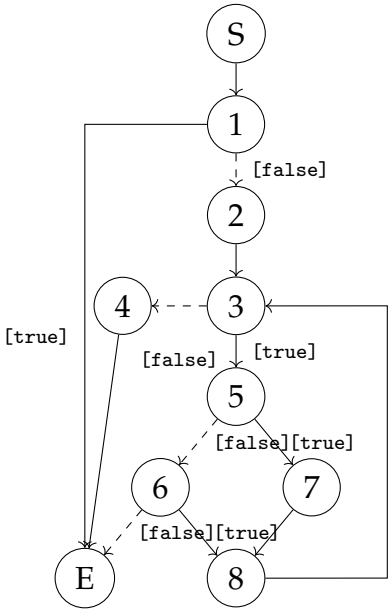
Lösungsvorschlag Kontrollflussgraph

Das Kriterium all-uses ist das Hauptkriterium des datenflussorientierten Testens, denn es testet den kompletten prädikativen und berechnenden Datenfluss. Konkret: von jedem Knoten mit einem globalen def(x) einer Variable x existiert ein definitions-freier Pfad bzgl. x (def-clear(x)) zu jedem erreichbaren Knoten mit einem c-use(x) oder p-use(x)).

66116 / 2017 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 1

Gegeben Sei folgende Methode und ihr Kontrollflussgraph:

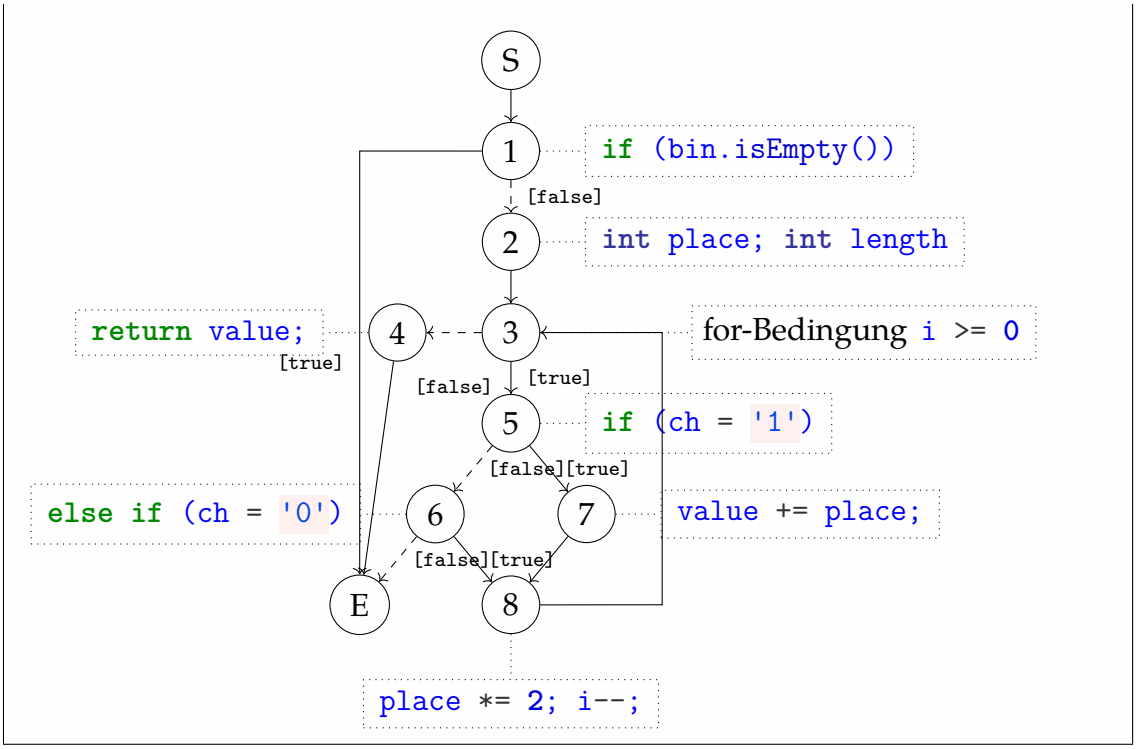
```java
int binToInt(String bin) {
  if (bin.isEmpty())
    return -1;
  int place = 1, value = 0;
  int length = bin.length() -
↪ 1;
  for (int i = length; i >= 0;
↪ --i) {
    char ch = bin.charAt(i);
    if (ch == '1') {
      value += place;
    } else if (ch == '0') {
      // do nothing
    } else {
      return -1;
    }
    place *= 2;
  }
  return value;
}
```



Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/sosy/ab_7/Aufgabe5.java`

(a) Geben Sie je einen Repräsentanten aller Pfadklassen im Kontrollflussgraphen an, die zum Erzielen einer vollständigen

Lösungsvorschlag



(i) Verzweigungsüberdeckung

Lösungsvorschlag

p1 (Pfad 1) S 1 E
p2 S 1 2 3 4 E
p3 S 1 2 3 5 7 8 3 5 6 8 3 5 6 E

(ii) Schleife-Inneres-Überdeckung

Lösungsvorschlag

Äußere Pfade (äußere Pfade): (ohne Ausführung der Wiederholung)

p1 S 1 E
p2 S 1 2 3 4 E

Grenzpfade (boundary test) (alle Pfade, die die Wiederholung betreten, aber nicht wiederholen; innerhalb des Schleifenrumpfes alle Pfade!)

p4 S 1 2 3 5 6 E

Innere Pfade (interior test) (alle Pfade mit *einer Wiederholung des Schleifenrumpfes*; innerhalb des Schleifenrumpfes wieder alle Pfade!)

p5 S 1 2 3 5 7 8 3 5 7 8 3 4 E
p6 S 1 2 3 5 7 8 3 5 6 8 3 4 E
p7 S 1 2 3 5 6 8 3 5 6 8 3 4 E
p8 S 1 2 3 5 6 8 3 5 7 8 3 4 E
p9 S 1 2 3 5 7 8 3 5 7 8 3 5 6 E

p10 = p3 S 1 2 3 5 7 8 3 5 6 8 3 5 6 E
p11 S 1 2 3 5 6 8 3 5 6 8 3 5 6 E
p12 S 1 2 3 5 6 8 3 5 7 8 3 5 6 E

mit minimaler Testfallanzahl genügen würden.

(b) Welche der vorangehend ermittelten Pfade sind mittels Testfälle tatsächlich überdeckbar („feasible“)? Falls der Pfad ausführbar ist, geben Sie bitte den Testfall an, andernfalls begründen Sie kurz, weshalb der Pfad nicht überdeckbar ist.

Lösungsvorschlag

Erweitere Methode, die die Knotennamen ausgibt:

```
public static final String RESET = "\u001B[0m";
public static final String ROT = "\u001B[31m";
public static final String GRÜN = "\u001B[32m";

static int binToIntLog(String bin) {
  System.out.println("\nInput: " + bin);
  System.out.print("S");
  System.out.print(1);
  if (bin.isEmpty()) {
    System.out.print("E");
    System.out.println("\nOutput: " + -1);
    return -1;
  }
  System.out.print(2);
  int place = 1, value = 0;
  int length = bin.length() - 1;
  System.out.print(3);
  for (int i = length; i >= 0; --i) {
    char ch = bin.charAt(i);
    System.out.print(5);
    if (ch == '1') {
      System.out.print(ROT + 7 + RESET);
      value += place;
    } else {
      System.out.print(GRÜN + 6 + RESET);
      if (ch == '0') {
        // do nothing
      } else {
        System.out.print("E");
        System.out.println("\nOutput: " + -1);
        return -1;
      }
    }
    System.out.print(8);
    place *= 2;
    System.out.print(3);
  }
  System.out.print(4);
  System.out.print("E");
  System.out.println("\nOutput: " + value);
  return value;
```

Zyklomatische Komplexität nach Mc-Cabe

```
  }

  public static void main(String[] args) {
    binToIntLog(""); // p1
    binToIntLog("??"); // p2 not feasible
    binToIntLog("x01"); // p3
    binToIntLog("x"); // p4

    binToIntLog("11"); // p5
    binToIntLog("01"); // p6
    binToIntLog("00"); // p7
    binToIntLog("10"); // p8

    binToIntLog("x11"); // p9
    binToIntLog("x01"); // p10
    binToIntLog("x00"); // p11
    binToIntLog("x10"); // p12
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/aufgaben/sosy/ab_7/Aufgabe5.java`

Alle mit Ausnahme von p2.

p2 ist nicht überdeckbar. Passiert ein Wert der Variable `bin` die erste if-Verzweigung, dann hat der Wert eine Länge größer 0 und betritt deshalb die Wiederholung mit fester Anzahl.

p1	S 1 E	`binToInt("");`
p2	S 1 2 3 4 E	not feasible
p3	S 1 2 3 5 7 8 3 5 6 8 3 5 6 E	`binToInt("x01");`
p4	S 1 2 3 5 6 E	`binToInt("x");`
p5	S 1 2 3 5 7 8 3 5 7 8 3 4 E	`binToInt("11");`
p6	S 1 2 3 5 7 8 3 5 6 8 3 4 E	`binToInt("01");`
p7	S 1 2 3 5 6 8 3 5 6 8 3 4 E	`binToInt("00");`
p8	S 1 2 3 5 6 8 3 5 7 8 3 4 E	`binToInt("10");`
p9	S 1 2 3 5 7 8 3 5 7 8 3 5 6 E	`binToInt("x11");`
p10 = p3	S 1 2 3 5 7 8 3 5 6 8 3 5 6 E	`binToInt("x01");`
p11	S 1 2 3 5 6 8 3 5 6 8 3 5 6 E	`binToInt("x00");`
p12	S 1 2 3 5 6 8 3 5 7 8 3 5 6 E	`binToInt("x10");`

(c) Bestimmen Sie anhand des Kontrollflussgraphen die maximale Anzahl linear unabhängiger Programmpfade, also die zyklomatische Komplexität nach Mc-Cabe.

Lösungsvorschlag

Datenfluss-annotierter Kontrollflussgraph

Binärverzweigungen 4

Knoten 10

Kanten 13

Anhand der Binärverzweigungen:

$$\begin{aligned} M &= b + p \\ &= 4 + 1 \\ &= 5 \end{aligned}$$

oder durch Anzahl Kanten e und Knoten n

$$\begin{aligned} M &= e - n + 2p \\ &= 13 - 10 + 2 \cdot 1 \\ &= 5 \end{aligned}$$

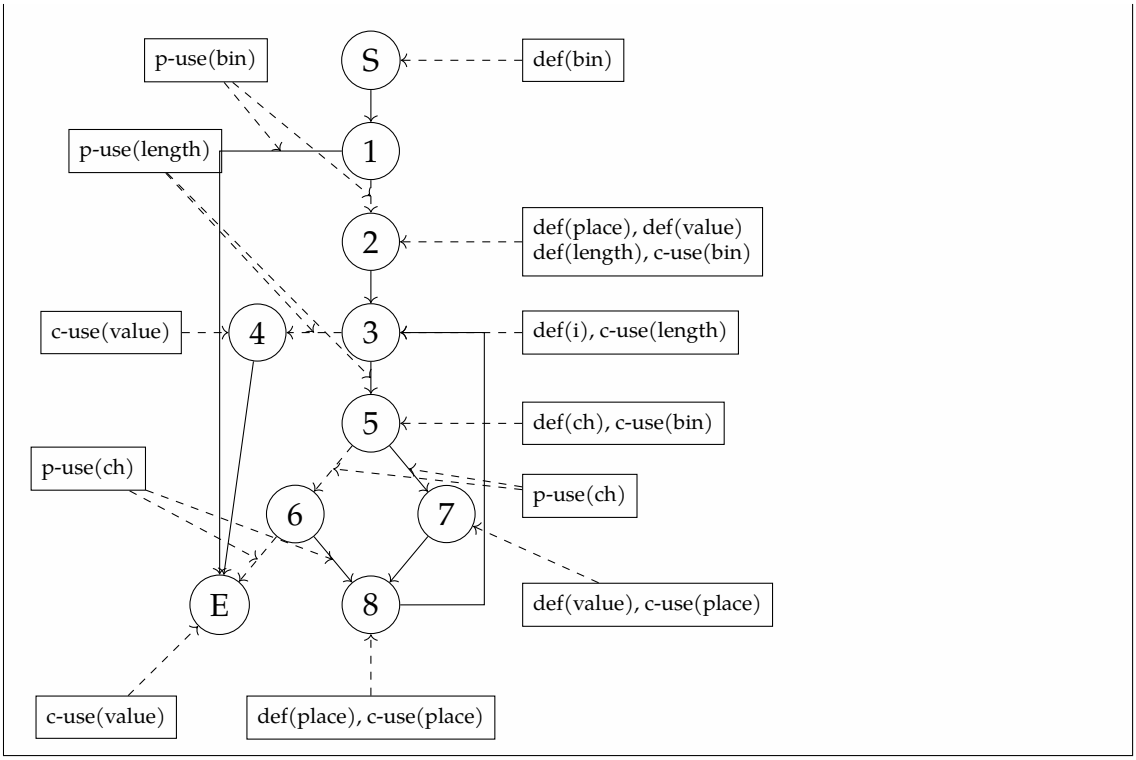
(d) Kann für dieses Modul eine 100%-ige Pfadüberdeckung erzielt werden? Begründen Sie kurz Ihre Antwort.

Lösungsvorschlag

Nein, da **p2** nicht überdeckbar ist.

(e) Geben Sie zu jedem Knoten die jeweilige Datenfluss-Annotation (`defs` bzw. `uses`) für jede betroffene Variable in der zeitlichen Reihenfolge ihres AuftreteS zur Laufzeit an.

Lösungsvorschlag



66116 / 2017 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 4

Sie dürfen im Folgenden davon ausgehen, dass keinerlei Under- oder Overflows auftreten.

Gegeben sei die folgende Methode mit Vorbedingung $P := x \geq 0 \wedge y \geq 0$ und Nachbedingung $Q := x \cdot y = z$.

```
int mul (int x , int y) {
  /* P */
  int z = 0, i = 0;
  while (i++ != x)
    z += y;
  /* Q */
  return z;
}
```

Betrachten Sie dazu die folgenden drei Prädikate:

- $I_1 := z + i \cdot y = x \cdot y$
- $I_2 :=$ false
- $I_3 := z + (x - i) \cdot y = x \cdot y$

(a) Beweisen Sie formal für jedes der drei Prädikate, ob es unmittelbar vor Betreten der Schleife in `mul` gilt oder nicht.

Lösungsvorschlag

$$\begin{aligned}\text{wp}(\texttt{"Code vor der Schleife"}, I_1) &\equiv \text{wp}(\texttt{"int z = 0, i = 0;"}, z + i \cdot y = x \cdot y)\\ &\equiv \text{wp}(\texttt{""}, 0 + 0 \cdot y = x \cdot y)\\ &\equiv 0 = x \cdot y\\ &\equiv \text{falsch}\end{aligned}$$

$$\begin{aligned}\text{wp}(\texttt{"Code vor der Schleife"}, I_2) &\equiv \text{wp}(\texttt{"int z = 0, i = 0;"}, \text{false})\\ &\equiv \text{wp}(\texttt{""}, \text{false})\\ &\equiv \text{false}\\ &\equiv \text{falsch}\end{aligned}$$

$$\begin{aligned}\text{wp}(\texttt{"Code vor der Schleife"}, I_3) &\equiv \text{wp}(\texttt{"int z = 0, i = 0;"}, z + (x - i) \cdot y = x \cdot y)\\ &\equiv \text{wp}(\texttt{""}, 0 + (x - 0) \cdot y = x \cdot y)\\ &\equiv x \cdot y = x \cdot y\\ &\equiv \text{wahr}\end{aligned}$$

(b) Weisen Sie formal nach, welche der drei Prädikate Invarianten des Schleifenrumpfs in `mul` sind oder welche nicht.

Lösungsvorschlag

Für den Nachweis muss der Code etwas umformuliert werden:

```
int mul (int x , int y) {
  /* P */
  int z = 0, i = 0;
  while (i != x) {
    i = i + 1;
    z = z + y;
  }
  /* Q */
  return z;
}
```

$$\begin{aligned}\text{wp}(\texttt{"Code Schleife"}, I_1 \wedge i \neq x) &\equiv \text{wp}(\texttt{"i = i + 1; z = z + y;"}, z + i \cdot y = x \cdot y \wedge i \neq x)\\ &\equiv \text{wp}(\texttt{"i = i + 1;"}, z + y + i \cdot y = x \cdot y \wedge i \neq x)\\ &\equiv \text{wp}(\texttt{""}, z + y + (i+1) \cdot y = x \cdot y \wedge i + 1 \neq x)\\ &\equiv z + y + (i+1) \cdot y = x \cdot y \wedge i + 1 \neq x\\ &\equiv z + i \cdot y + 2 \cdot y = x \cdot y \wedge i + 1 \neq x\\ &\equiv \text{falsch} \wedge i + 1 \neq x\\ &\equiv \text{falsch}\end{aligned}$$

$$\begin{aligned}\text{wp}(\texttt{"Code Schleife"}, I_2 \wedge i \neq x) &\equiv \text{wp}(\texttt{"i = i + 1; z = z + y;"}, \text{false} \wedge i \neq x)\\ &\equiv \text{wp}(\texttt{""}, \text{false} \wedge i \neq x)\\ &\equiv \text{falsch} \wedge i \neq x\\ &\equiv \text{falsch}\end{aligned}$$

$$\begin{aligned}\text{wp}(\texttt{"Code Schleife"}, I_3 \wedge i \neq x) &\equiv \text{wp}(\texttt{"i = i + 1; z = z + y;"}, z + (x - i) \cdot y = x \cdot y \wedge i \neq x)\\ &\equiv \text{wp}(\texttt{"i = i + 1;"}, z + y + (x - i) \cdot y = x \cdot y \wedge i \neq x)\\ &\equiv \text{wp}(\texttt{""}, z + y + (x - i + 1) \cdot y = x \cdot y \wedge i + 1 \neq x)\\ &\equiv z + y + x \cdot y - i \cdot y + y = x \cdot y \wedge i + 1 \neq x\\ &\equiv z + 2 \cdot y + x \cdot y - i \cdot y = x \cdot y \wedge i + 1 \neq x\\ &\equiv \text{wahr}\end{aligned}$$

(c) Beweisen Sie formal, aus welchen der drei Prädikate die Nachbedingung gefolgert werden darf bzw. nicht gefolgert werden kann.

Lösungsvorschlag

$I_1 := z + i \cdot y = x \cdot y$ $I_2 := \text{false}$ $I_3 := z + (x - i) \cdot y = x \cdot y$

$$\begin{aligned}\text{wp}(\texttt{"Code nach Schleife"}, I_1 \wedge i = x) &\equiv \text{wp}(\texttt{""}, z + i \cdot y = x \cdot y \wedge i = x)\\ &\equiv z + i \cdot y = x \cdot y \wedge i = x\\ &\equiv z + x \cdot y = x \cdot y\\ &\neq Q\end{aligned}$$

$$\begin{aligned}\text{wp}(\texttt{"Code nach Schleife"}, I_2 \land i = x) &\equiv \text{wp}(\texttt{""}, \text{false} \land i = x)\\ &\equiv \text{false} \land i = x\\ &\equiv \text{falsch}\\ &\neq Q\end{aligned}$$

$$\begin{aligned}\text{wp}(\texttt{"Code nach Schleife"}, I_3 \land i = x) &\equiv \text{wp}(\texttt{""}, z + (x - i) \cdot y = x \cdot y \land i = x)\\ &\equiv z + (x - i) \cdot y = x \cdot y \land i = x\\ &\equiv z + (x - x) \cdot y = x \cdot y\\ &\equiv z + 0 \cdot y = x \cdot y\\ &\equiv z + 0 = x \cdot y\\ &\equiv z = x \cdot y\\ &\equiv Q\end{aligned}$$

(d) Skizzieren Sie den Beweis der totalen Korrektheit der Methode `mul`. Zeigen Sie dazu auch die Terminierung der Methode.

Lösungsvorschlag

Aus den Teilaufgaben folgt der Beweis der partiellen Korrektheit mit Hilfe der Invariante i_3. i steigt streng monoton von 0 an so lange gilt $i \neq x$. $i = x$ ist die Abbruchbedingung für die bedingte Wiederholung. Dann terminiert die Methode. Die Methode `mul` ist also total korrekt.

66116 / 2017 / Herbst / Thema 1 / Teilaufgabe 2 / Aufgabe 4

Die folgende Seite enthält Software-Quellcode, der einen Algorithmus zur binären Suche implementiert. Dieser ist durch Inspektion zu überprüfen. Im Folgenden sind die Regeln der Inspektion angegeben.

RM1	(Dokumentation)	Jede Quellcode-Datei beginnt mit einem Kommentar, der den Klassennamen, Versionsinformationen, Datum und Urheberrechtsangaben enthält.
RM2	(Dokumentation)	Jede Methode wird kommentiert. Der Kommentar enthält eine vollständige Beschreibung der Signatur so wie eine Design-by-Contract-Spezifikation.
RM3	(Dokumentation)	Deklarationen von Variablen werden kommentiert.
RM4	(Dokumentation)	Jede Kontrollstruktur wird kommentiert.
RM5	(Formatierung)	Zwischen einem Schlüsselwort und einer Klammer steht ein Leerzeichen.
RM6	(Formatierung)	Zwischen binären Operatoren und den Operanden stehen Leerzeichen.
RM7	(Programmierung)	Variablen werden in der Anweisung initialisiert, in der sie auch deklariert werden.
RM8	(Bezeichner)	Klassennamen werden groß geschrieben, Variablennamen klein.

```
/**
 * BinarySearch.java
 *
 * Eine Implementierung der "Binaere Suche"
 * mit einem iterativen Algorithmus
 */
class BinarySearch {

  /**
   * BinaereSuche
   * a: Eingabefeld
   * item: zusuchendesElement
   * returnValue: der Index des zu suchenden Elements oder -1
   *
   * Vorbedingung:
   * a.length > 0
   * a ist ein linear geordnetes Feld:
   * For all k: (1 <= k < a.length) ==> (a[k-1] <=a [k])
   *
   * Nachbedingung:
   * Wenn item in a, dann gibt es ein k mit a[k] == item und returnValue == k
   * Genau dann wenn returnValue == -1 gibt es kein k mit 0 <= k < a.length
   * und a[k]==item.
   */
  public static int binarySearch(float a[], float item) {

    int End;  // exklusiver Index fuer das Ende des
              // zudurchsuchenden Teils des Arrays
```

```
    int start = 1; // inklusiver Index fuer den Anfang der Suche
    End = a.length;

    // Die Schleife wird verlassen, wenn keine der beiden Haelften das
    // Element enthaelt.
    while(start < End) {

      // Teilung des Arrays in zwei Haelften
      // untere Haelfte: [0,mid[
      // obere Haelfte: ]mid,End[
      int mid = (start + End) / 2;

      if (item > a[mid]) {
        // Ausschluss der oberen Haelfte
        start = mid + 1;
      } else if(item < a[mid]) {
        // Ausschluss der unteren Haelfte
        End = mid-1;
      } else {
        // Das gesuchte Element wird zurueckgegeben
        return (mid);
      }
    } // end of while

    // Bei Misserfolg der Suche wird -1 zurueckgegeben
    return (-1);
  }
}
```

(a) Überprüfen Sie durch Inspektion, ob die obigen Regeln für den Quellcode eingehalten wurden. Erstellen Sie eine Liste mit allen Verletzungen der Regeln. Geben Sie für jede Verletzung einer Regel die Zeilennummer, Regelnummer und Kommentar an, z. B. (07, RM4, while nicht kommentiert). Schreiben Sie nicht in den Quellcode.

Lösungsvorschlag

Zeile	Regel	Kommentar
3-8	RM1	Fehlen von Versionsinformationen, Datum und Urheberrechtsangaben
11-26	RM2	Fehlen der Invariante in der Design-by-Contract-Spezifikation
36,46	RM5	Fehlen des Leerzeichens vor der Klammer
48	RM6	Um einen binären (zweistellige) Operator handelt es sich im Code-Beispiel um den Subtraktionsoperator: `mid-1`. Hier fehlen die geforderten Leerzeichen.
32	RM7	Die Variable `End` wird in Zeile 32 deklariert, aber erst in Zeile initialisiert `End = a.length;`
32	RM8	Die Variable `End` muss klein geschrieben werden.

(b) Entspricht die Methode `binarySearch` ihrer Spezifikation, die durch Vor-und Nachbedingungen angeben ist? Geben Sie gegebenenfalls Korrekturen der Methode an.

Lösungsvorschlag

Korrektur der Vorbedingung

Die Vorbedingung ist nicht erfüllt, da weder die Länge des Feldes `a` noch die Reihenfolge der Feldeinträge geprüft wurden.

```
if (a.length <= 0) {
  return -1;
}

for (int i = 0; i < a.length; i ++) {
  if ( a[i] > a[i + 1]) {
    return -1;
  }
}
```

Korrektur der Nachbedingung

`int start` muss mit `0` initialisiert werden, da sonst `a[0]` vernachlässigt wird.

(c) Beschreiben alle Kommentare ab Zeile 24 die Semantik des Codes korrekt? Geben Sie zu jedem falschen Kommentar einen korrigierten Kommentar mit Zeilennummer an.

Lösungsvorschlag

Zeile	Kommentar im Code	Korrektur
34-35	`// Die Schleife wird v⌋ erlassen, wenn keine der beiden Haelften das Elemen⌋ t enthaelt.`	`// Die Schleife wird v⌋ erlassen, wenn keine der beiden Haelften das El⌋ ement enthaelt oder das Element gefunden wurde.`
44	`// Ausschluss der oberen Haelfte`	`// Ausschluss der unteren Haelfte`
47	`// Ausschluss der unteren Haelfte`	`// Ausschluss der oberen Haelfte`
50	`// Das gesuchte Element wird zurueckgegeben`	`// Der Index des gesuchten Elements wird zurueckgeb⌋ en`

(d) Geben Sie den Kontrollflussgraphen für die Methode `binarySearch` an.

(e) Geben Sie maximal drei Testfälle für die Methode `binarySearch` an, die insgesamt eine vollständige Anweisungsüberdeckung leisten.

Lösungsvorschlag

Die gegebene Methode: `binarySearch(a[], item)`

Testfall

(i) Testfall: `a[] = {1, 2, 3}, item = 4`

(ii) Testfall: `a[] = {1, 2, 3}, item = 2`

66116 / 2019 / Frühjahr / Thema 1 / Teilaufgabe 2 / Aufgabe 3

(a) Erläutern Sie kurz, was man unter der Methode der testgetriebenen Entwicklung versteht.

Lösungsvorschlag

Bei der testgetriebenen Entwicklung erstellt der/die ProgrammiererIn Softwaretests konsequent vor den zu testenden Komponenten.

(b) Geben Sie für obige Aufgabenstellung (Abarbeitung der Aufträge durch den Roboter) einen Testfall für eine typische Situation an (d. h. das Einsammeln von 4 Objekten an unterschiedlichen Positionen). Spezifizieren Sie als Input alle für die Abarbeitung eines Auftrages relevanten Eingabe- und Klassen-Parameter sowie den vollständigen und korrekten Output.

66116 / 2019 / Frühjahr / Thema 2 / Teilaufgabe 2 / Aufgabe 1

Testen

Bewerten Sie die folgenden Aussagen und nehmen Sie jeweils Stellung.

(a) Tests zuerst

In test-getriebener Softwareentwicklung wird der Test vor der zu testenden Funktion programmiert.

Lösungsvorschlag

Bei der testgetriebenen Entwicklung erstellt der/die ProgrammiererIn Softwaretests konsequent vor den zu testenden Komponenten.

(b) Komponententests

Komponententests sind immer White-Box-Tests und nie Black-Box-Tests.

Lösungsvorschlag

Falsch: Komponententests (anderes Wort für Unit- oder Modul-Tests) können beides sein. [a]

[a] `https://qastack.com.de/software/362746/what-is-black-box-unit-testing`

(c) Akzeptanztests

Akzeptanz - resp. Funktionstests sind immer Black-Box-Tests und nie White-Box-Tests.

Lösungsvorschlag

In systems engineering, it may involve black-box testing performed on a system

Acceptance testing in extreme programming: Acceptance tests are black-box system tests.

a

[a] `https://en.wikipedia.org/wiki/Acceptance_testing`

(d) Fehler

Ein fehlgeschlagener Test und ein Testausführungsfehler bezeichnen denselben Sachverhalt.

Lösungsvorschlag

Falsch:

Fehler (Error)

Der Software-Test konnte durchgeführt werden, das ermittelte Ist-Ergebnis und das vorgegebene Soll-Ergebnis weichen jedoch voneinander ab. Ein derartiger Fehler liegt z. B. dann vor, wenn ein Funktionsaufruf einen abweichenden Wert berechnet.

Fehlschlag (Failure)

Während der Testdurchführung wurde ein Systemversagen festgestellt. Ein

Fehlschlag liegt z. B. dann vor, wenn das zu testende System einen Programmabsturz verursacht und hierdurch erst gar kein Ist-Ergebnis ermittelt werden konnte.

Das Misslingen kann als Ursache einen Fehler (Error) oder ein falsches Ergebnis (Failure) haben, die beide per Exception signalisiert werden. Der Unterschied zwischen den beiden Begriffen liegt darin, dass Failures erwartet werden, während Errors eher unerwartet auftreten. [a]

White-Box-Testing

[a] https://de.wikipedia.org/wiki/JUnit

(e) Test Suiten

Tests können hierarchisch strukturiert werden, so dass mit einem Befehl das gesamte zu testende System getestet werden kann.

Lösungsvorschlag

Richtig:

test suite (englisch „Testsammlung“, aus französisch suite Folge, Verkettung) bezeichnet eine organisierte Sammlung von Werkzeugen zum Testen technischer Apparate und Vorgänge. [a]

[a] https://de.wikipedia.org/wiki/Testsuite

66116 / 2019 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 3

Eine Dezimalzahl hat ein optionales Vorzeichen, dem eine nichtleere Sequenz von Dezimalziffern folgt. Der anschließende gebrochene Anteil ist optional und besteht aus einem Dezimalpunkt, gefolgt von einer nichtleeren Sequenz von Dezimalziffern.

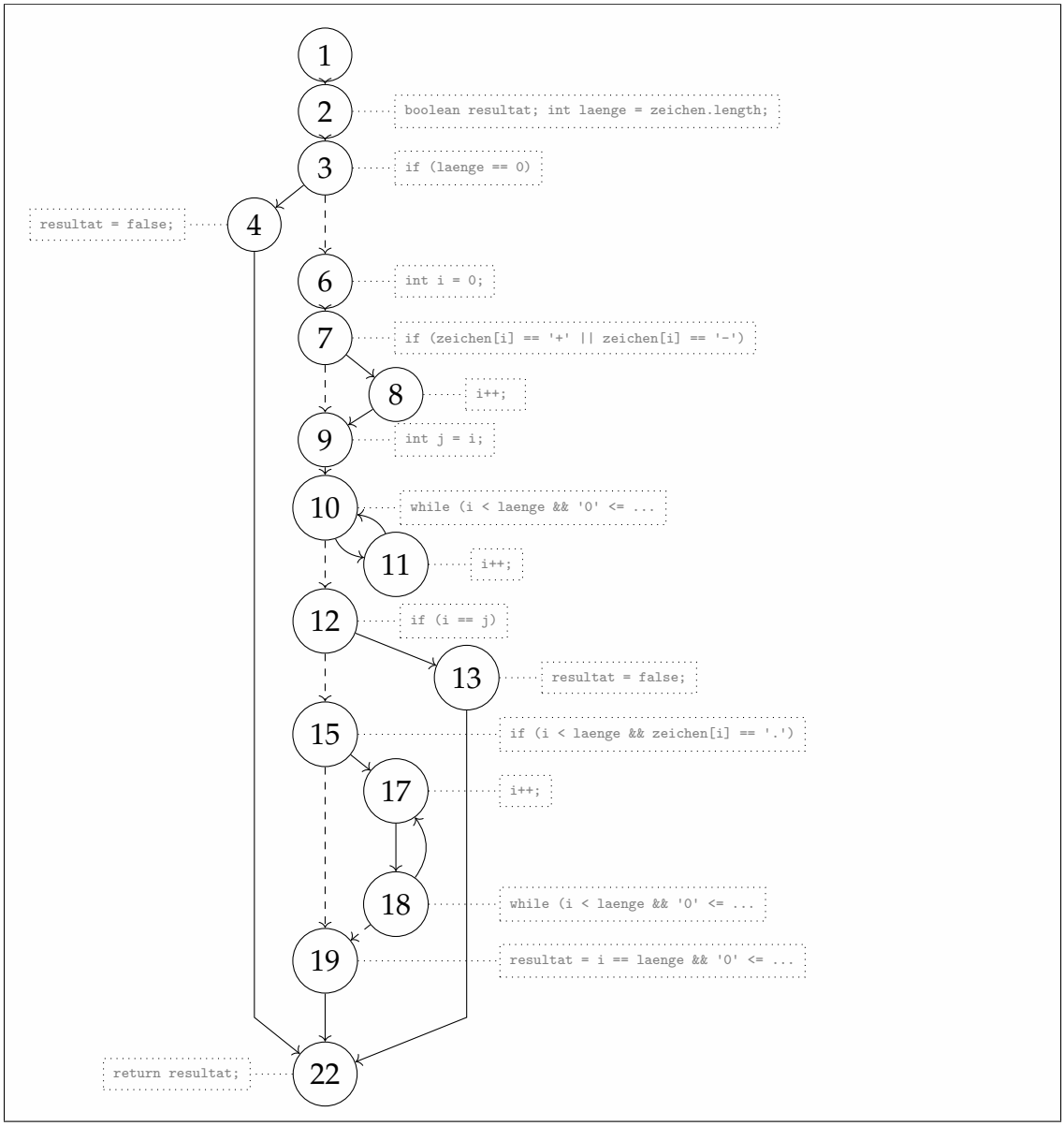
Die folgende Java-Methode erkennt, ob eine Zeichenfolge eine Dezimalzahl ist:

```
public static boolean istDezimalzahl(char[] zeichen) { // 1
  boolean resultat; int laenge = zeichen.length; // 2
  if (laenge == 0) // 3
    resultat = false; // 4
  else { // 5
    int i = 0; // 6
    if (zeichen[i] == '+' || zeichen[i] == '-') // 7
      i++; // 8
    int j = i; // 9
    while (i < laenge && '0' <= zeichen[i] && zeichen[i] <= '9') // 10
      i++; // 11
    if (i == j) // 12
      resultat = false; // 13
    else { // 14
      if (i < laenge && zeichen[i] == '.') // 15
        do // 16
          i++; // 17
        while (i < laenge && '0' <= zeichen[i] && zeichen[i] <= '9'); // 18
      resultat = i == laenge && '0' <= zeichen[i - 1] && zeichen[i - 1] <= '9';
↪  // 19
    } // 20
```

```
    } // 21
    return resultat; // 22
  }
```

(a) Konstruieren Sie zu dieser Methode einen Kontrollflußgraphen. Markieren Sie dessen Knoten mit Zeilennummern des Quelltexts.

Lösungsvorschlag



(b) Geben Sie eine minimale Testmenge an, die das Kriterium der Knotenüberdeckung erfüllt. Geben Sie für jeden Testfall den durchlaufenen Pfad in der Notation $1 \to 2 \to \ldots$ an.

Lösungsvorschlag

- „“:
  (1) - (2) - (3) - (4) - (22)
- „1“:

① - ② - ③ - ⑥ - ⑦ - ⑨ - ⑩ - ⑪ - ⑩ - ⑫ - ⑮ - ⑲ - ㉒

- „+1.0“:
  ① - ② - ③ - ⑥ - ⑦ - ⑧ - ⑨ - ⑩ - ⑪ - ⑩ - ⑫ - ⑮ - ⑰ - ⑱ - ⑲ - ㉒
- „x“:
  ① - ② - ③ - ⑥ - ⑦ - ⑨ - ⑩ - ⑫ - ⑬ - ㉒

(c) Verfahren Sie wie in b) für das Kriterium der Kantenüberdeckung.

Lösungsvorschlag

text

(d) Wie stehen die Kriterien der Knoten- und Kantenüberdeckung zueinander in Beziehung? Begründen Sie Ihre Anwort.

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

Lösungsvorschlag

Die Kantenüberdeckung fordert, dass jede *Kante* des Kontrollflussgraphen von mindestens einem Testfall durchlaufen werden muss. Um das Kriterium zu erfüllen, müssen die Testfälle so gewählt werden, dass jede Verzweigungsbedingung mindestens *einmal wahr* und mindestens *einmal falsch* wird. Da hierdurch alle Knoten ebenfalls mindestens einmal besucht werden müssen, ist die *Anweisungsüberdeckung* in der Zweigüberdeckung *vollständig enthalten*.

66116 / 2020 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 1

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

Lösungsvorschlag

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.

Lösungsvorschlag

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.

Lösungsvorschlag

Schritt 0: Schleifeninvariante I finden

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

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

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

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

Lösungsvorschlag

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

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

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!

wp-Kalkül
White-Box-Testing

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

```
int i = -1;
// (1)
int j = 0;
// (2)
while (i == -1 && j < n) // (3)
{ // (4)
  if (a[j] == m) {
    // (5)
    i = j;
    // (6)
  } else {
    // (7)
    j = j + 1;
    // (8)
  }
  // (9)
}
```

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

$Q \equiv P \land (i = -1 \land \forall\, 0 \leq k < n \colon a_k \neq m) \lor (i \geq 0 \land a_i = m)$

Lösungsvorschlag

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

66116 / 2020 / Herbst / Thema 1 / Teilaufgabe 1 / Aufgabe 4

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.

Kontrollflussgraph

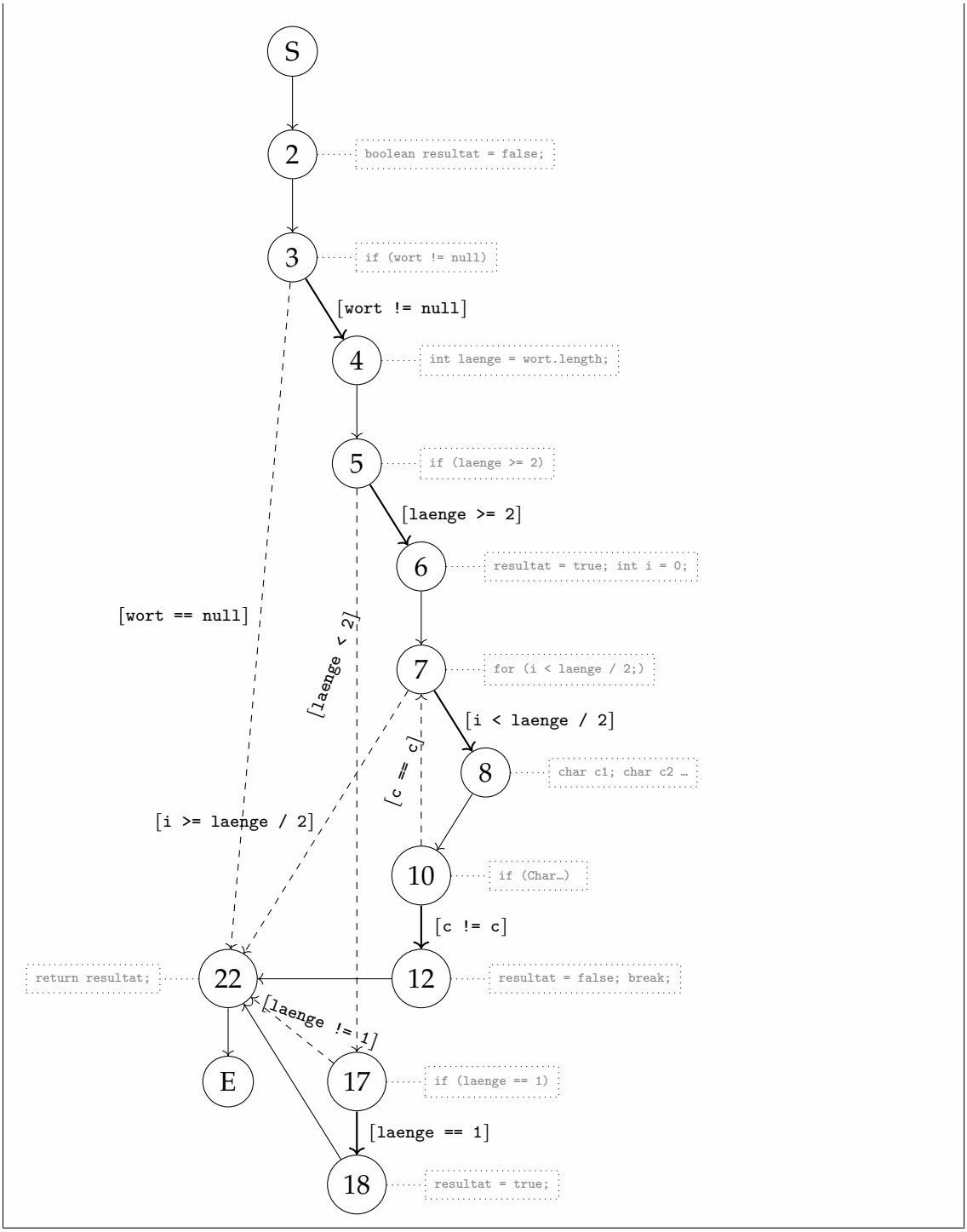
Folgende *Java-Methode* prüft, ob das übergebene Zeichen-Array ein Wortpalindrom darstellt:

```
public static boolean istWortpalindrom(char[] wort) { // 1
  boolean resultat = false; // 2
  if (wort != null) { // 3
    int laenge = wort.length; // 4
    if (laenge >= 2) { // 5
      resultat = true; // 6
      for (int i = 0; i < laenge / 2; ++i) { // 7
        char c1 = wort[i]; // 8
        char c2 = wort[laenge - 1 - i]; // 9
        if (Character.toLowerCase(c1) != Character.toLowerCase(c2)) // 10
        { // 11
          resultat = false; // 12
          break; // 13
        } // 14
      } // 15
    } else { // 16
      if (laenge == 1) { // 17
        resultat = true; // 18
      } // 19
    } // 20
  } // 21
  return resultat; // 22
} // 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.

Lösungsvorschlag

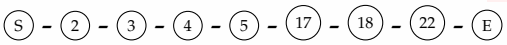


(b) Geben Sie eine *minimale Testmenge* an, die das Kriterium der Anweisungsüberdeckung erfüllt.

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

Lösungsvorschlag

```
isWortpalindrom(new char[] { 'a' }):
```



`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.

Lösungsvorschlag

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).

Lösungsvorschlag

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. Testen

Lösungsvorschlag

Kante 7 nach 22

66116 / 2020 / Herbst / Thema 2 / Teilaufgabe 1 / Aufgabe 6

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

```
/** Return the Greatest Common Divisor of two integer values. */

public int gcd(int a, int b) {
  if (a < 0 || b < 0) {
  return gcd(-b, a);

  }
  while (a != b) {
    if (a>b){
      a = a - b;
    } else {
      b = b % a;
    }
  }
  return 3;
}
```

(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.

Sonstige

SOSY