Fallsituationen für die Softwareentwicklung mit JAVA

Die folgende Aufgabensammlung versucht anhand von einfachen Fallbeispielen mit überwiegend betriebswirtschaftlichem Hintergrund die Vorgehensweise bei der objektorientierten Softwareentwicklung zu vermitteln. Grundlage ist hierbei ein **Phasenmodel**l, welches den Softwareentwicklungsprozess in die Phasen:

Problemstellung – Analyse – Design – Codierung –Test – Implementierung

einteilt. Schwerpunkt bilden die Phasen Analyse und Design. Dabei werden die **UML-Diagramme Anwendungsfalldiagramm – Klassendiagramm** – **Objektdiagramm** und **Sequenzdiagramm** verwendet. Die Codierung erfolgt in der Programmiersprache **JAVA.**
Die notwendigen Programmiergrundlagen finden sich in meinem Skript „Skript für den Informatikunterricht an der Max-Weber-Schule“, welches unter www.pellatz.de heruntergeladen werden kann.

Grundsätzliche Hinweise zur Codierung:

Es wird davon ausgegangen, dass passive Klassen und aktive Klassen (Anwendungen mit einer main-Methode) grundsätzlich voneinander getrennt sind. Die Gestaltung von Attributen und Methoden sollte bekannt sein. Attribute werden grundsätzlich gekapselt und besitzen get- und set-Methoden. Die Verwendung von logischen Strukturen (Verzweigung, Mehrfachauswahl und Wiederholungen) sollte bekannt sein. Es sollte eine Möglichkeit, Tastatureingaben zu realisieren (z.B. Scanner) geläufig sein. Außerdem sollte das Anlegen und Verarbeiten von Arrays (oder anderer Container) bekannt sein.

Fallsituationen mit Musterlösung

Inhalt

1. Renovieren
2. Firmenumsatz – OPAL
3. Kunde und Betreuer
4. Auftragsstatistik
5. Wer liefert was?
7. Kursverwaltung
8. Wer sitzt wo?
9. Pizzaservice
10. Japan Wines
11. Supermarkt
13. Autovermietung

Fallsituationen ohne Musterlösung

12. Lagerverwaltung
6. Geldautomat
14. Platzreservierung
15. Bibliothek
16. Photovoltaik

Fallbeispiel 1 Renovieren

Peter Merz hat sich eine neue Wohnung gemietet. Vor dem Einzug ist noch viel zu tun.
Der Fußboden muss neu verlegt werden und die Wände müssen gestrichen werden. Für die Berechnung der notwendigen Materialmengen muss Peter Merz die Grundfläche jedes einzelnen Zimmers sowie die Wandfläche jedes Zimmers berechnen.

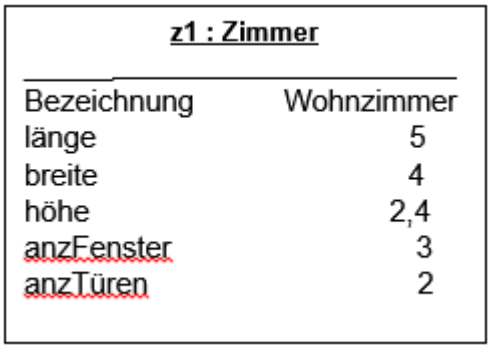
Aufgabenstellung:

Lösen Sie die genannten Anwendungsfälle mit einer objektorientierten Software.
Alle Zimmer sind rechteckig, Türen und Fenster haben einheitliche Maße.

a) Erstellen Sie zunächst ein **Anwendungsfalldiagramm** (use-case-diagramm).

b) Entwerfen Sie den Aufbau einer Klasse Zimmer mit den notwendigen Attributen und Methoden in einem **Klassendiagramm**. Alle Attribute sind gekapselt.

c) Codieren Sie die Klasse Zimmer auf der Grundlage des abgebildeten Objektdiagramms und erstellen Sie eine Applikation zum Testen der Anwendungsfälle. Erzeugen Sie mindestens zwei Instanzen der Klasse Zimmer.



Erweiterung 1:

d) Codieren Sie einen **Konstruktor**, mit dem alle notwendigen Attribute initialisiert werden können und nutzen Sie diesen in der Applikation.

Erweiterung 2:

e) Für die Anschaffung der Wandfarbe sowie für den Fußbodenbelag soll die Summe aller Bodenflächen und Wandflächen berechnet werden. Zu diesem Zweck müssen alle Instanzen der Klasse Zimmer in einem **Array** gespeichert werden, welches dann mit einem Wiederholungskonstrukt verarbeitet werden kann.
Ändern Sie dazu die Erzeugung der Objekte und berechnen Sie die jeweiligen Summen mit einer for-Schleife.

Fallbeispiel 2 Firmenumsatz - OPAL

Der Autohersteller OPAL hat zwei Werke in *Dortmund* und in *Essen*, die zentral von der Firma in Köln verwaltet werden. Jedes Werk stellt einen bestimmten Fahrzeugtyp her. In Dortmund wird der *Saphir* hergestellt, in Essen der *Smaragd*. Für jeden Fahrzeugtyp werden der Stückpreis und die Verkaufsmenge gespeichert.

1. Erstellen Sie ein Klassendiagramm mit Assoziationen und Kardinalitäten

2. Legen Sie die notwendigen Klassen in Java mit folgenden Attributen an:
   Firma: **Name** (String)
   Werk: **Standort** (String),
   Produkt: **Bezeichnung** (String), **Preis** (double), **Verkaufsmenge** (int)
   (zunächst werden keine Verweisattribute benötigt)

   Für jedes Attribut soll eine get- und eine set-Methode existieren.

3. Erzeugen Sie alle genannten Objekte und initialisieren Sie die Attribute mit den genannten Daten sowie mit folgenden Werten:

   Saphir: **Preis** *20 000 €*, **Verkaufsmenge** *15 0*
   Smaragd: **Preis** *16 500 €*, **Verkaufsmenge** *220*

   Die Initialisierungen können auch mit einem Konstruktor erfolgen. Lassen Sie alle Daten am Bildschirm ausgeben (verwenden Sie dazu evtl. die toString-Methode). Die Applikation soll die Bezeichnung **`Umsatz1`** tragen.

4. Es wird ein drittes Werk in *Siegen* eröffnet, in dem LKWs vom Typ *Mammut* hergestellt werden. Die Klasse **LKW** ist von der Klasse Produkt abgeleitet und erhält ein zusätzliches Attribut mit der Bezeichnung **Zuladung** (int)**.** Erzeugen Sie diese Klasse und ein Objekt von dieser Klasse sowie ein neues Werk in der Applikation **`Umsatz2`**.

5. Wir erzeugen die Assoziationen zwischen den Klassen Werk und Produkt. Die Klasse Werk bekommt ein zusätzliches Attribut **meinProdukt** vom Typ **Produkt**, Codieren Sie dieses Attribut und erstellen Sie dazu eine get- und set-Methode. Initialisieren Sie die Attribute in der Applikation **`Umsatz3`**.

6. Die Klasse Werk soll eine Methode bekommen, mit der der Umsatz des Werkes berechnet wird. Das Ergebnis wird in dem neuen Attribut **WerksUmsatz** (double) gespeichert. Realisieren Sie diese Methode ***bWerksUmsatz*** und lassen sie alle Werksumsätze für die drei Werke in der Applikation **`Umsatz4`** berechnen.

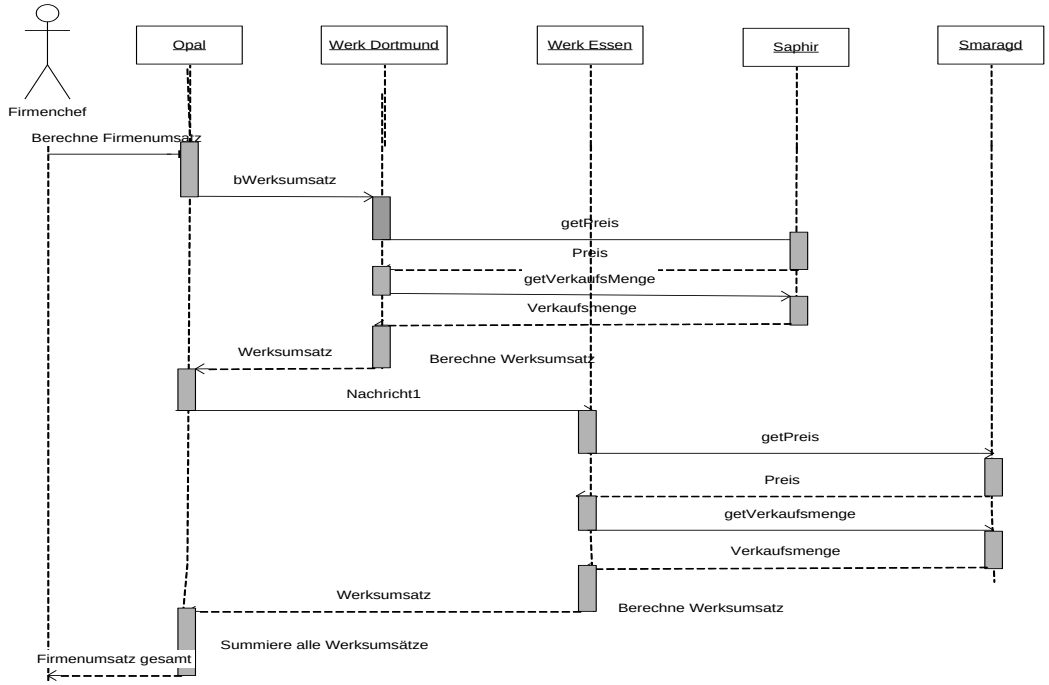
7. Der Firmeninhaber möchte eine Übersicht über den Umsatz der drei Werke.

   Realisieren Sie den Anwendungsfall ***BerechneFirmenumsatz*** als Methode der Klasse **Firma** anhand des vorliegenden **Sequenzdiagrammes** in der Applikation **`Umsatz5`**. Es soll dabei folgende Ausgabe (je nach Zahlen) am Bildschirm erzeugt werden:

```
Werk Dortmund: Produkt Saphir, Umsatz 3000000 Euro
Werk Essen: Produkt Smaragd, Umsatz 36300000 Euro
Werk Siegen: Produkt Mammut, Umsatz 2400000 Euro

Gesamtumsatz der Firma OPAL  9030000 Euro
```

(Sie können die Methode bWerksUmsatz in der Klasse Firma ohne entsprechendes Verweisattribut direkt mit der Objektbezeichnung aufrufen. Preis und Verkaufsmenge werden aber mit dem entsprechenden Verweisattribut (siehe Aufg. 5) angesprochen.)

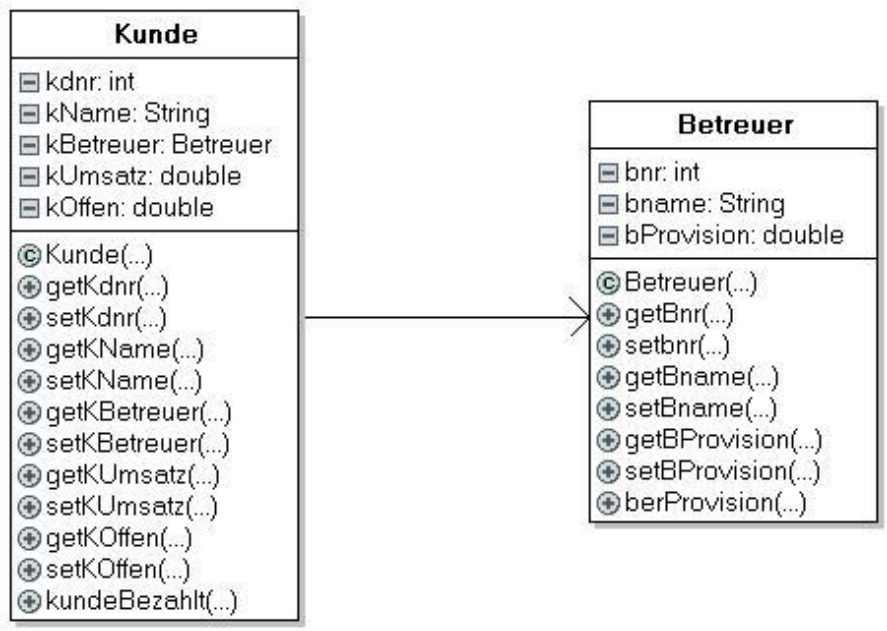
Sequenzdiagramm für die Methode ***berechneFirmenumsatz***
(zur Vereinfachung ist hier das Werk Siegen nicht mit aufgeführt



Fallbeispiel 3 Kunde und Betreuer

Jeder Kunde eines Unternehmens wird von einem Kundenbetreuer betreut. Wenn der Kunde einen Auftrag erteilt, bekommt der Betreuer eine Provision von 5% der vom Kunden bezahlten Auftragssumme. Bei Autragserteilung wird eine Rechnung erstellt und das Attribut Offene Posten beim Kunden um den Rechnungsbetrag erhöht. Jeder Zahlungseingang des Kunden vermindert die Offenen Posten und führt zu einer Provisionsberechnung beim Betreuer. Eine Software soll die Anwendungsfälle **Kunde erteilt Auftrag**, **Kunde bezahlt Rechnung** und **Betreuerprovision anzeigen** realisieren.

1. Erstellen Sie die Klassen Kunde und Betreuer nach folgendem Klassendiagramm:

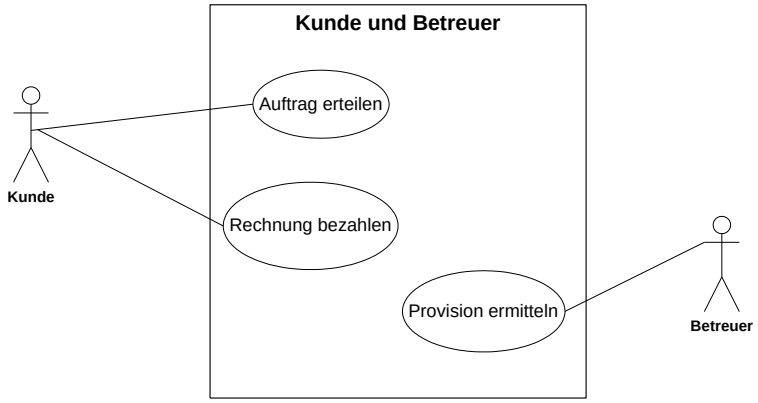


Die Methoden ***kundeBezahlt*** und ***berProvision*** können zunächst noch weggelassen werden. Erzeugen Sie mindestens jeweils drei Kunden und Betreuer über einen Konstruktor. Die Attributwerte für **kUmsatz**, **kOffen** und **bProvision** werden mit dem Wert 0 initialisiert. Damit Kunden oder Betreuer ausgewählt werden können, sollen die Objekte der beiden Klassen in einem Array gespeichert werden. Die Syntax dafür lautet:

```
//Arrays für Klassenobjekte erzeugen
Kunde[] k = new Kunde[5];
Betreuer [] b = new Betreuer[5];
//Objekte erzeugen und intialiseren
k[0]= new Kunde(101,"Meier");
b[0]= new Betreuer(1001,"Hans");
k[0].setKBetreuer(b[0]);
```

2. Realisieren Sie in die Applikation **`Kundenbetreuung1`** den Anwendungsfall **KundeErteiltAuftrag**.

3. Erstellen Sie die Applikation **`Kundenbetreuung2`**, die die folgenden Anwendungsfälle realisiert.
   Die Anwendungsfälle sollen mit Hilfe eines Auswahlmenüs über ein switch..case-Konstrukt angesteuert werden.



Sofern die einzelnen Anwendungsfälle über eine eigene Prozedur aufgerufen werden, müssen die Arrays für die Klassen Kunden und Betreuer übergeben werden.

Aufruf: auftrag(k, b) → **Prozedurkopf**: public static void auftrag(Kunde[ ] k, Betreuer [ ] b)

Fallbeispiel 4 Auftragsstatistik

In einem Unternehmen soll die Auftragsbearbeitung mit einem neuen Programm erledigt werden. In einem Klassendiagramm soll zunächst die Struktur des Systems modelliert werden. Die Grundlage dazu liefern folgende Informationen:

Wenn ein Kunde einen Auftrag erteilt, werden die einzelnen Auftragspositionen erfasst. Bei jeder Auftragsposition wird anhand der Produktdaten die Lieferfähigkeit überprüft. Nach Abschluss der Auftragserfassung wird eine Rechnung generiert.

Aufträge werden mit Auftragsnummer, Datum und Auftragssumme erfasst.
Für Kunden werden die Kundennummer, Name und Anschrift gespeichert.
Auftragspositionen beinhalten die Positionsnummer, Artikelnummer und Menge.
Die Produktdaten speichern Artikelnummer, Artikelbezeichnung, Bestand und Preis.
Rechnungen beinhalten die Rechnungsnummer und das Rechnungsdatum.

a) Erstellen Sie zunächst das **Klassendiagramm.**
b) Wie werden die **Assoziationen** zwischen den Klassen realisiert?
c) Stellen Sie den Anwendungsfall „Auftragsstatistik erstellen“ in einem **Sequenzdiagramm** dar
   Als Ergebnis der Auftragsstatistik soll täglich eine Liste mit den Aufträgen des Tages erstellt werden nach folgendem Muster:

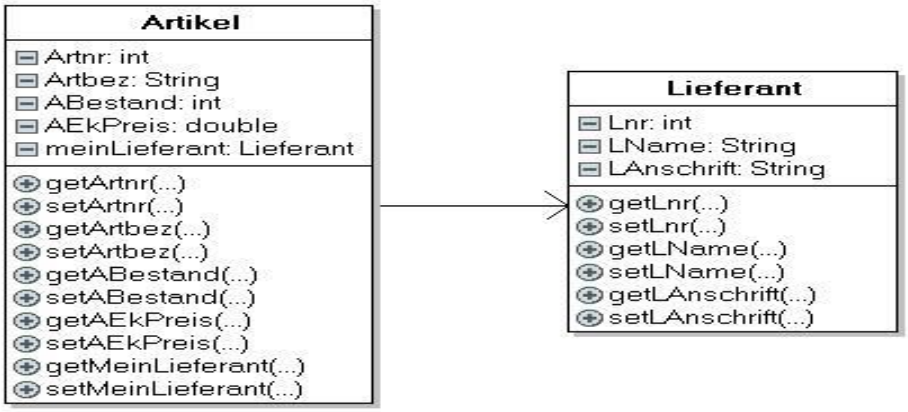
Auftragsnummer	Kundennummer	Kundenname	Auftragssumme
080703001	2007	Meier KG	559,50
080703002	2003	Rinn und Keil	1300,00
080703003	2013	Loose GmbH	712,50
080703004	2003	Rinn und Keil	79,95

Fallbeispiel 5 Wer liefert was?

Jeder Artikel eines Handelsunternehmens hat genau einen Lieferanten.
Die Geschäftsleitung möchte folgende Anwendungsfälle realisieren:

- Ausgabe einer tabellarischen Artikelliste und einer tabellarischen Lieferantenliste.
- Ausgabe des gesamten Lagerwertes aller Artikel.
- Anzeige des Lieferanten anhand einer eingegebenen Artikelnummer

1. Erstellen Sie ein Klassendiagramm
2. Erstellen Sie ein Use-case-Diagramm für die genannten Anwendungsfälle
3. Codieren Sie die beiden Klassen anhand des vorliegenden Klassendiagramms.



4. Erstellen Sie die Applikationen **Artikelliste** und **Lieferantenliste**.
   Erzeugen Sie die Instanzen für die Klasse Artikel und die Klasse Lieferanten über ein Array. Es sollen mindestens 5 Artikel und 3 Lieferanten über einen Konstruktor erzeugt werden.

5. Erstellen Sie die Applikation **LagerwertErmitteln.** Die Auswertung der Objekte erfolgt mit einer for-Schleife. Zeichnen Sie ein **Sequenzdiagramm** für diese Applikation.

6. Erstellen Sie die Applikation **werLiefertWas**. Damit sollen anhand der Artikelnummer, die über die Tastatur eingegeben wird, die Daten des Lieferanten angezeigt werden.

   Die Eingabe soll so lange wiederholt werden, bis als Artikelnummer 0 eingegeben wird. Falls eine Artikelnummer nicht vorhanden ist, soll die Meldung „Artikel nicht vorhanden“ angezeigt werden.

Fallbeispiel 6 Geldautomat

Eine Bank betreut Kunden und verwaltet Konten. Jeder Kunde hat genau ein Konto. Die Kunden wollen am Geldautomat ihren Kontostand abfragen, Geld einzahlen und Geld abheben. Realisieren Sie das System Geldautomat.

1. Erstellen Sie ein Anwendungsfalldiagramm
2. Erstellen Sie ein Klassendiagramm mit Assoziationen und Kardinalitäten ohne Attribute und Methoden.
3. Erstellen Sie für die Klassen Kunde und Konto jeweils ein detailliertes Klassendiagramm mit folgenden Attributen:
   Kunde: Kundennummer (kdnr), Name (kName), Anrede (kAnrede), Pin (kPin).
   Konto: Kontonummer (ktNr), Kontostand (ktStand).
   (wir verzichten auf die IBAN, um nicht immer 22 Stellen eingeben zu müssen.)
   Die Klasse Konto beinhaltet die Methoden *einzahle*n und *auszahlen*.
4. Erstellen Sie den Quellcode für die beiden Klassen einschließlich des Konstruktors.
5. Realisieren Sie die Assoziation zwischen dem Kunden und seinem Konto.
6. Erstellen Sie eine Applikation als Prototypen. Erzeugen Sie mindestens drei Kunden und drei Konten und testen sie die genannten Anwendungsfälle. Für die Kunden verwenden Sie ein Array.
7. Erstellen Sie den Geldautomaten als Applikation mit folgenden Anforderungen:
   Der Kunde identifiziert sich mit seiner pin und wird daraufhin mit Anrede und Name begrüßt. Danach erhält er ein Auswahlmenü (switch…case) mit den drei genannten Anwendungsfällen sowie einer Option zum Beenden der Anwendung.

Erweiterung

8. Jeder Kunde hat zwei Konten: Ein Sparkonto und ein Girokonto. Das Sparkonto hat einen eigenen Zinssatz als zusätzliches Attribut, das Girokonto erhält einen Dispo als zusätzliches Attribut, der die Höhe der möglichen Kontoüberziehung angibt. Erstellen Sie dafür eine Vererbungsbeziehung.
9. Die Methode *auszahlen* funktioniert beim Sparkonto und beim Girokonto unterschiedlich. Das Sparkonto kann nicht überzogen werden, das Girokonto kann bis zur Höhe des Dispos überzogen werden. Realisieren Sie in beiden Klassen zusätzlich diese polymorphe Methode.

Wir haben den Geldautomat auch mit einer graphischen Oberfläche versehen. Der Kunde identifiziert sich mit seiner Kundennummer. (Bankkarte wäre auch möglich, wenn man ein Kartenlesegerät hat). Die nächste Seite zeigt die Benutzeroberfläche.


Geldautomat
MWS - BANK
Kundennummer
100
Eingabe
Guten Tag, Herr Wagner
Bitte wählen Sie
Abfrage
Auszahlen
Einzahlen
Pin eingeben
••••
Eingabe
Sie haben Zugriff auf das Konto 200
Aktueller Kontostand: 1.150,00 Euro
Nächster Kunde

Fallbeispiel 7 Kursverwaltung

Ein Fortbildungsinstitut möchte seine Software zur Kursverwaltung auf die objektorientierte Programmierung umstellen. Zu diesem Zweck soll zunächst folgender Sachverhalt als Klassendiagramm modelliert werden:

Für die Teilnehmer eines Kurses werden die Teilnehmernummer, der Name, die Anschrift und der Status (beschäftigt, Schüler/Student bzw. arbeitslos) gespeichert. Jeder Teilnehmer kann sich für ein oder mehrere Kurse anmelden. Für jeden Kurs werden dessen Nummer, die Bezeichnung, das Datum sowie die Kursgebühr gespeichert. An einem Kurs können nicht mehr als 20 Teilnehmer teilnehmen.
Jeder Kurs wird von einem Kursleiter angeboten. Ein Kursleiter kann mehrere Kurse anbieten. Für den Kursleiter werden Name, Anschrift und Firma gespeichert.
Jeder Teilnehmer hat genau ein Konto. Im Konto werden Kontonummer, und die bezahlte Kursgebühr gespeichert.

1. Erstellen Sie das Klassendiagramm mit den Klassenbezeichnungen, den Attributen und den Assoziationen Realisieren Sie für die Kursleiter und die Teilnehmer eine Generalisierungsbeziehung. Get- und set-Methoden für die Attribute müssen nicht dargestellt werden.

2. Welche möglichen Anwendungsfälle sind für das beschriebene System denkbar? Stellen Sie diese in einem Anwendungsfalldiagramm dar.

3. Zu Beginn eines Kurses erhält jeder Kursleiter eine Kursliste, in der alle angemeldeten Teilnehmer aufgelistet sind. Beschreiben Sie, wie die Erstellung dieser Liste realisiert werden kann und ob dafür möglicherweise eine zusätzliche Assoziation erforderlich ist.

Fallbeispiel 8 Wer sitzt wo?

In einer Sportarena gibt es Blöcke, Reihen und Sitzplätze. Besucher können freie Plätze reservieren. Es soll ein System entwickelt werden, welches die freien Plätze anzeigt und namentliche Platzreservierungen vornehmen kann.
Realisieren Sie diese Anwendung in einem Beispiel mit mindestens 30 Plätzen (2 Blöcke mit je 3 Sitzreihen und 5 Plätzen pro Sitzreihe.

Weitere Überlegungen

Es können die Klassen **Platz** und **Besucher** angelegt werden.
Attribute für die Klasse Platz sind die Platzid, Block, Reihe und Platznummer sowie ein Verweisattribut für den Besucher.
So lange der Platz nicht belegt ist, ist das Verweisattribut mit frei zu bezeichnen.

Für den Besucher können Name, Vorname und Anschriftdaten erfasst werden.
Der Name genügt aber für die Funktionalität. Außerdem ein Verweisattribut auf den gebuchten Platz. Dabei kann ein Besucher auch mehrere Plätze gebucht haben (Erweiterung)

Die Applilakation kann in Form eines Menüs dargestellt werden.
1. Alle Plätze anzeigen
2. Freie Plätze anzeigen
3. Platz buchen
4. Besucher und seine gebuchten Plätze anzeigen.
5. Buchung stornieren.

Fallbeispiel 9 Pizzaservice

Der Pizza-Service Bella Italia bietet seinen Kunden Pizzen in zwei Größen mit 12 unterschiedlichen Belägen an, von denen der Kunde beliebig viele für eine Pizza auswählen kann. Es gibt drei Preiskategorien für Beläge:

Preiskategorie 1	Paprika
	Peperoni
	Knoblauch
	Oliven
	Zwiebeln
Preiskategorie 2	Salami
	Schinken
	extra Käse
	Ananas
Preiskategorie 3	Gorgonzola
	Muscheln
	Shrimps

	28 cm	32 cm
Grundpizza	4,00	4,50
Preiskategorie 1	0,50	0,70
Preiskategorie 2	1,00	1,30
Preiskategorie 3	1,20	1,60

Die Grundpizza enthält immer Tomaten und Käse

Ihre Aufgabe ist es, ein Softwaresystem zu entwickeln, das einen Pizzamitarbeiter bei der Annahme der Bestellungen gut unterstützt. Dazu soll es alle für diesen Zweck notwendigen Daten verarbeiten können. Am Ende der Bestellung soll der Preis für die Pizza angezeigt werden.
(Diese Aufgabe war eine von 5 Aufgaben des Bundeswettebewerbs für Informatik 2009.)

1. Beschreiben Sie, wie der Vorgang der Bestellannahme mit Hilfe des Systems ablaufen soll. Wählen Sie dazu eine geeignete graphische Darstellung.

2. Entwerfen Sie für das System eine Benutzerschnittstelle und stellen Sie diese graphisch dar.

3. Entwickeln Sie ein Datenmodell für das System in Form eines Klassendiagramms

4. Realisieren Sie die Kernfunktionalität des Systems

Fallbeispiel 10 Japan Wines

Um die Geschäftsprozesse der Japan Wines Inc. (JWI) mit Hilfe einer neuen Software zu optimieren wurden anhand von Interviews die folgenden 28 Aussagen gesammelt:

1	Japan Wines (JWI) ist ein Wein-Vertriebszentrum.
2	Das Geschäft der JWI besteht darin, den Lagerbestand zu verwalten und Produkte an Einzelhändler entsprechend deren Bestellungen zu liefern.
3	Alle Bestellungen, die an einem Tag einlaufen, werden am nächsten Tag bearbeitet.
4	Das Lager wird täglich überprüft und aufgrund des festgestellten Bedarfs werden Bestellungen an Weinproduzenten erteilt.
5	Das Zentrum ist nicht zuständig für Abrechnungen mit den Einzelhändlern oder Weinproduzenten.
6	Es folgt eine genauere Beschreibung der Geschäftsabläufe:
7	Das Zentrum empfängt Bestellungen der Einzelhändler telephonisch von 9 – 17 Uhr.
8	Das Bestellformular enthält das Datum, die Bestellnummer, den Namen und die Adresse des Einzelhändlers sowie eine Auflistung der bestellten Ware mit Bezeichnung und Menge.
9	Eine Bestellung kann aus mehreren Bestellposten bestehen. Diese beziehen sich auf einzelne Produkte.
10	Jeder Bestellposten wird auf dem Formular in einer Zeile notiert.
11	Wenn das Zentrum eine Bestellung erhält, wird umgehend der Lagerbestand für jeden Bestellposten geprüft.
12	Das Zentrum nimmt anerkannte Reklamationen von den Einzelhändlern zurück.
13	Wenn das Zentrum eine Bestellung entgegennimmt, wird jeder Bestellposten in eine der beiden folgenden Karteien eingeordnet: a() Liefer-Kartei, (b) Warte-Kartei
14	Falls das Zentrum für einen Bestellposten genügend Ware auf Lager hat, dann wird dieser in die Liefer-Kartei eingetragen und der Lagerbestand wird entsprechend korrigiert.
15	Andernfalls wird der Bestellposten in die Warte-Kartei eingetragen.
16	Täglich nach 17 Uhr werden die notwendigen Bestellungen an die Weinproduzenten sowie die Auslieferungs-Anweisungen wie folgt erstellt:
17	Bestellungen an Weinproduzenten werden für solche Produkte erteilt, deren Mengen unter dem Minimal-Lagerbestand gesunken sind. Dabei wird so viel bestellt, dass der entsprechende Maximal-Lagerbestand erreicht wird.
18	Das Formular für die Bestellungen enthält das Datum, die Bestellnummer, den Produzenten mit Name und Adresse sowie eine Auflistung der bestellten Ware mit Bezeichnung und Menge.
19	Minimal- und Maximal-Lagerbestände werden für jedes Produkt definiert.
20	Das Zentrum erstellt Lieferscheine für jeden Lieferwagen. Dabei werden den Lieferwagen die Bestellposten aus der Liefer-Kartei entsprechend ihrer Fahrtziele zugeordnet.
21	Der Lieferschein enthält das Datum, Name und Adresse des Einzelhändlers, Liefer- und Bestellnummer, Lieferwagen-Nummer sowie eine Auflistung der gelieferten Posten
22	Am nächsten Morgen lädt jeder Lieferwagen Produkte aus dem Lagerhaus gemäß dem Lieferschein und liefert diese aus.
23	Nach erfolgter Auslieferung werden die Lieferscheine mit dem Vermerk ‚ausgeliefert' oder ‚Retoure' zurückgebracht.

24	Produkte mit dem Vermerk ‚Retoure' werden ins Lagerhaus zurückgebracht.
25	Das Zentrum löscht alle ausgelieferten Bestellposten aus der Liefer-Kartei.
26	Bestellposten mit dem Vermerk ‚Retoure' verbleiben in der Liefer-Kartei und werden am Folgetag in die Lieferscheine übernommen.
27	Frische Produkte von den Weinproduzenten treffen zwischen 10 und 16 Uhr ein.
28	Nach Eintreffen der frischen Produkte werden diese den Bestellposten der Warte-Kartei zugeordnet. Alle nun lieferbaren Posten werden von dort in die Liefer-Kartei übertragen. Der Rest der frischen Ware wird dem Lagerbestand zugeführt.

1. Ermitteln Sie auf der Grundlage der 28 informellen Aussagen die **Anwendungsfälle**. Idealerweise lassen sich **acht Arbeitsabläufe** feststellen, aus denen sich die Anwendungsfälle ergeben.
   In Absprache mit dem Projektleiter kann die Bearbeitung von Reklamationen bei der Analyse und Modellierung ausgeklammert werden.
   Als Personen, die mit den Anwendungsfällen agieren (**Akteure)** wurden der Sachbearbeiter im Büro und der Lagerverwalter festgelegt.
   Zwischen den Akteuren wurde in einem Anwendungsfall eine Kommunikations-beziehung festgestellt. Diese wird als bidirektional angesehen und wird durch jeweils eine Verbindungslinie zu beiden Akteuren dargestellt.
   Der Anwendungsfall ‚Lagerbestand prüfen' wird von den Anwendungsfällen ‚Bestellung bearbeiten' und ‚Fehlbestände nachbestellen' benutzt (extend-Beziehung).

   Erstellen sie aufgrund dieser Überlegungen ein **Anwendungsfalldiagramm.**

2. Formulieren Sie zu jedem der genannten Anwendungsfälle eine **verbale Beschreibung**.
   (textuelle Beschreibung der Anwendungsfälle)

3. Ermitteln Sie anhand der informellen Aussagen und der textuellen Beschreibungen der Anwendungsfälle geeignete **Klassenkandidaten**. Idealerweise erhalten Sie auf diese Art acht Klassenkandidaten.

4. Überlegen Sie, ob es zwischen einzelnen Klassen Gemeinsamkeiten gibt, so dass sich Oberklassen bilden lassen, von denen diese Klassen abgeleitet werden. (**Vererbungs-beziehungen**).

5. Erstellen Sie nun einen ersten Entwurf für ein **Klassendiagramm** mit **Assoziationen** und **Kardinalitäten**.

6. Überlegen Sie nun, welche möglichen **Attribute** und **Methoden** die einzelnen Klassen haben sollten, damit die genannten Anwendungsfälle realisiert werden können. Stellen Sie die Ergebnisse in **detaillierten Klassendiagrammen** dar. Bedenken Sie aber, dass diese Entwürfe vermutlich im Laufe der weiteren gedanklichen Auseinandersetzung noch mehrmals geändert werden.

7. Als nächstes wenden wir uns der dynamischen Struktur des Systems zu, indem wir zu den Methoden jeder Klasse ein **Sequenzdiagramm** zeichnen. Diese Tätigkeit steht im Wechselspiel mit der nachträglichen Anpassung der Attribute und Methoden.
   Danach können wir in die **Codierungsphase** einsteigen.
   Diesen Punkt (Arbeitsauftrag 7) brauchen Sie aber nicht mehr durchzuführen.

Fallbeispiel 11 Supermarkt

Eine Supermarkkette verwaltet ihre Supermärkte von einer Zentrale aus. Zur Abrechnung der Einkäufe eines Kunden stehen einem Supermarkt mehrere Kassen bereit. Eine Kasse speichert am Morgen die bereitgestellte Bargeldeinlage und den aktuellen Kassenbestand. Wenn ein Kunde seine Einkäufe bezahlt, wird der Kassenbestand um diesen Betrag erhöht. Nach Geschäftsschluss wird von der Zentrale die gesamte Tageseinnahme aller Märkte ermittelt.

1. Erstellen Sie ein Klassendiagramm für die für das Abrechnungssystem relevanten Klassen.

2. Erstellen Sie ein Objektdiagramm für die Zentrale, zwei Supermärkte und je zwei Kassen pro Markt. Die Attribute der der Objekte haben folgende Ausprägung:
   Zentrale z1: tagesEinnahmenGesamt = noch offen
   Supermarkt s1: name = WMarkt, tagesEinnahme = noch offen
   Kasse k1: bezeichnung = Kasse 1, einlage = 250, kassenstand = 1700
   Kasse k2: bezeichung = Kasse 2, einlage = 125, kassenstand = 1279
   Supermarkt s2: name = KaufGut, tagesEinahme = noch offen
   Kasse k3: bezeichnung = Kasse 3, einlage = 89, kassenstand = 873
   Kasse k4: bezeichung = Kasse 4, einlage = 120, kassenstand = 1079

3. Überlegen Sie, welche Anwendungsfälle sich aus der beschriebenen Situation ergeben und stellen Sie diese in einem Anwendungsfalldiagramm dar. Berücksichtigen Sie dabei, dass der Kunde nicht als Akteur gelten kann, da der Kunde nur mittelbar über den Kassierer mit der Software interagiert.

4. Formulieren Sie die Methoden der einzelnen Klassen, welche zum Anwendungsfall ‚Gesamte Tageseinnahmen ermitteln' erforderlich sind und stellen Sie diesen Anwendungsfall in einem Sequenzdiagramm (Interaktionsdiagramm) dar.

Fallbeispiel 12 Lagerverwaltung

Diese Aufgabe erweitert die Fallsituation 5 (Wer liefert was?).

Als Vorarbeit zur Erstellung eines Warenwirtschaftssystems soll zunächst die Lagerverwaltung des Baumarktes „**Mach‘ es selber**“ erstellt werden. Es geht darum, dass sämtliche Verkaufsartikel des Baumarktes in einer Klasse „Artikel“ erfasst werden sollen. Damit sollen dann folgende Anwendungsfälle gelöst werden können: Ausgabe einer Liste aller Artikel, Anzeige des aktuellen Lagerwertes aller Artikel, Übersicht über alle Artikel deren Meldebestand erreicht oder unterschritten ist, Anzeige des Lieferanten für einen bestimmten Artikel, Liste aller Lieferanten, Erfassen des Warenausgangs (bei Verkauf), Erfassen des Wareneingangs, Nachbestellung von Artikeln.

Es sind folgende Zusatzinformationen vorhanden:

Jeder Artikel wird von genau einem „Lieferanten“ geliefert. Die Assoziation ist durch ein Verweisattribut in der Klasse Artikel zu realisieren. Für die Artikel werden nur die notwendigen Attribute erfasst: Artikelnummer, Artikelbezeichnung, aktueller Bestand, maximaler Bestand, Meldebestand, Einkaufspreis und der Bestellvermerk.
Die Lieferanten benötigen lediglich eine Lieferantennummer, Name und Anschrift.
Bei der Nachbestellung einzelner Artikel, deren Meldebestand erreicht oder unterschritten ist, wird der Anwendungsfall ‚Bestand prüfen‘ automatisch durchgeführt (include-Beziehung). Bei der Bestellung wird der Bestellvermerk auf true gesetzt, damit dieser Artikel nicht nochmals bestellt wird. Bei der Verbuchung des Wareneingangs wird dieser wieder zurückgesetzt.

Realisieren Sie die folgenden Aufgabenstellungen in der angegebenen Reihenfolge und Erstellen Sie eine Dokumentation

1. Erstellen Sie ein **Anwendungsfalldiagramm** für die genannten Anwendungsfälle.
2. Erstellen Sie **Klassendiagramm** für das komplette System und je ein detailliertes Klassendiagramm (mit Attributen und Methoden (außer get- und set) für die Klassen Artikel und Lieferant.
3. Erstellen Sie exemplarisch je ein **Objektdiagramm** für beide Klassen.
4. Programmieren Sie zunächst die Klasse **Artikel u**nd erzeugen Sie mindestens fünf Instanzen über einen **Konstruktor.** Zur Realisierung der Anwendungsfälle empfiehlt es sich, die Instanzen in einem **Container** (Array oder ArrayList) anzulegen.
5. Realisieren Sie die Ausgabe einer **Artikelliste**. Dazu könnte eine t**oString**-Methode verwendet werden.
6. Erstellen Sie ein **Sequenzdiagramm** für den Anwendungsfall „Lagerwert aller Artikel“.
7. Realisieren Sie die übrigen Anwendungsfälle. Es bleibt Ihnen überlassen, ob Sie die Anwendungfsälle mit mehreren Applikationsklassen oder einer menügesteuerten Applikation erstellen.

Für die Dokumentation sind alle verlangten Diagramme vorzulegen, der Quellcode der Klassen sowie der Quellcode der Applikationen. Erstellen Sie ein Deckblatt mit dem Thema der Aufgabenstellung sowie den Namen und Unterschriften der an der Erstellung beteiligten Personen.

Fallbeispiel 13. Autovermietung

Eine Autovermietung vermietet Fahrzeuge an Kunden. Wenn ein Kunde ein Auto mietet, hat dies eine Buchung zur Folge. Eine Buchung kann sich nur auf ein Fahrzeug beziehen.

Teil A:

1. Erstellen Sie für die dargestellte Situation ein Klassendiagramm mit Assoziationen und Kardinalitäten.

2. Programmieren Sie die Klasse Fahrzeug mit folgenden Attributen.
   Kennzeichen, Marke, Farbe, Verbrauch, kw, kmStand, maxTankinhalt, aktTankinhalt und **Mietsatz**.
   Alle Attribute sind als *private* und mit einem passenden Datentyp zu deklarieren.
   Außerdem ist für jedes Attribut eine ***set***- und eine ***get-Methode*** zu erstellen.

3. Legen Sie einen Konstruktor an, für die Initialisierung aller Attribute.

4. Erstellen Sie eine neue Applikation **`auto1`**, erzeugen Sie zwei Instanzen *a1* und *a2* und lassen Sie die Attributwerte der beiden Objekte wieder ausgeben.
   Die Ausgabe der Attributwerte kann mit einer Methode gelöst werden.

5. Die Klasse **Auto** soll zwei zusätzliche Methoden erhalten. Die Methode ***tanken*** mit dem Übergabeparameter *liter* und die Methode ***fahren*** mit dem Übergabeparameter *km*.

   Die Methode ***tanken*** verändert den aktuellen Tankinhalt. Dabei ist darauf zu achten, dass der maximale Tankinhalt nicht überschritten werden darf.
   Die Methode ***fahren*** verändert den kmStand sowie den aktuellen Tankinhalt.

6. Es soll eine neue Klasse **LKW** erzeugt werden, die eine von der Klasse Auto abgeleitete Klasse ist. Die Klasse LKW erhält als zusätzliches Attribute das **maxLadegewicht** und das **aktLadegewicht** (in kg). Die Methode ***beladen*** erhält als Übergabeparameter die Zuladung (in kg) und soll ein Beladen bis zur Höhe des maximalen Ladegewichts ermöglichen.
   Erzeugen Sie eine Instanz von dieser Klasse und lassen Sie die initialisierten Werte in der Applikation **`auto2`** wieder ausgeben. Testen Sie die Methode ***beladen*** aus.

Teil B:

In stark vereinfachter Form sollen nun die Anwendungsfälle **Kunde bucht Fahrzeug**, **Kunde gibt Fahrzeug zurück** und **Anzeige aller Buchungen** realisiert werden.

1. Erzeugen Sie die beiden Klassen **Kunde** und **Buchung**. Die Klasse Kunden hat die Attribute **kdnr** und **kName**, die Klasse Buchung die Attribute **bnr**, **Kennzeichen** und **Dauer**.

2. In der Applikation **`auto3`** wird zunächst der Anwendungsfall **Kunde bucht Fahrzeug** realisiert. Dabei werden 2 Instanzen der Klasse Auto und eine Instanz der Klasse Kunden erzeugt. In einem Bildschirmdialog wird der Kunde mit Namen begrüßt, die zur Verfügung stehenden Fahrzeuge angezeigt und der Kunde wird aufgefordert, das Kennzeichen des zu buchenden Fahrzeuges sowie die Mietdauer einzugeben. Der Kundenmethode ***buchen*** werden die Parameter Buchungsnummer, Kennzeichen und Mietdauer übergeben. Die Methode erzeugt sodann eine Instanz der Klasse **Buchung.**

Fallbeispiel 14 Platzreservierung

In einer Sportarena gibt es Blöcke, Reihen und Sitzplätze. Besucher können freie Plätze reservieren. Es soll ein System entwickelt werden, welches die freien Plätze anzeigt und namentliche Platzreservierungen vornehmen kann.
Realisieren Sie diese Anwendung in einem Beispiel mit mindestens 30 Plätzen
(2 Blöcke mit je 3 Sitzreihen und 5 Plätzen pro Sitzreihe.

Weitere Überlegungen

Es können die Klassen **Platz** und **Besucher** angelegt werden.
Attribute für die Klasse Platz sind die Platzid, Block, Reihe und Platznummer sowie ein Verweisattribut für den Besucher.
So lange der Platz nicht belegt ist, ist das Verweisattribut mit frei zu bezeichnen.

Für den Besucher können Name, Vorname und Anschriftdaten erfasst werden.
Der Name genügt aber für die Funktionalität. Außerdem ein Verweisattribut auf den gebuchten Platz. Dabei kann ein Besucher auch mehrere Plätze gebucht haben (Erweiterung)

Die Applilakation kann in Form eines Menüs dargestellt werden.
1. Alle Plätze anzeigen
2. Freie Plätze anzeigen
3. Platz buchen
4. Besucher und seine gebuchten Plätze anzeigen.
5. Buchung stornieren.

Fallbeispiel 15 Bibliothek

In einer Bibliothek werden Medien als Bücher und DVDs verliehen. Kunden (Entleiher)entleihen Medien und geben diese zurück. Die Bibliothekskraft (Verleiher) erfasst die Ausleihen bzw. die Rückgaben. Neue Kunden müssen zunächst erfasst werden (Kundenerfassung). Neue Medien werden ebenfalls erfasst (Medienerfassung). Die Bibliothekskraft kann anhand der Ausleihen feststellen, welche Medien entliehen sind und kann bei Überschreitung der Ausleihfrist eine Mahnung ausdrucken.

Folgende Informationen werden für die Kunden und die Medien benötigt:

Kunde: Kundennummer, Name, Anschrift

Medien: IdNr, Typ (Buch oder DVD), Titel, Verleihstatus,
Für Bücher: Erscheinungsjahr, Für DVDs: Spieldauer

Bei der Ausleihe werden folgende Daten erfasst:

Kundennummer, IdNr, Ausleihdatum (in der Form JJJJMMTT)

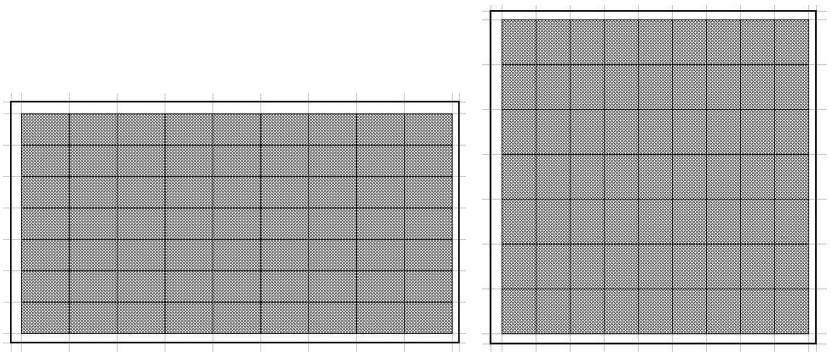
Weitere mögliche Anwendungsfälle können sein: Stammdaten für Kunden oder Medien verändern, Stammdaten löschen. Liste aller verliehenen Medien,

Über die Assoziationen zwischen den Klassen sollte diskutiert werden. Eventuell notwendige Verweisattribute können festgelegt werden.

Fallbeispiel 16 Photovoltaikanlage

Peter Cornelius ist Mitarbeiter der Firma *Sunpower*, die Photovoltaikanlagen installiert. Eine Photovoltaikanlage besteht aus einzelnen Solarmodulen, die auf das Dach montiert werden. Durch die Sonneneinstrahlung wird Strom erzeugt, der in das Stromnetz eingespeist wird und von den Stromgesellschaften nach dem Energieeinspeisungsgesetz in bestimmter Höhe vergütet werden muss. Je größer die Dachfläche, desto mehr Module können installiert werden und um so höher ist die Stromerzeugung.
Es gibt Module verschiedener Hersteller, die unterschiedliche Abmessungen haben.
Bei der Auswahl der geeigneten Module geht es in erster Linie darum, die Dachfläche optimal auszunutzen, so dass die Nutzfläche am größten ist.
Diese Frage kann Peter Cornelius im Kundengespräch nie sofort beantworten, da er für die Berechnung der optimalen Modulanordnung eine gewisse Zeit benötigt.

Ihre Aufgabe ist es, eine Software zu entwickeln, die je nach Dachabmessungen die geeigneten Module so auswählt, dass die nutzbare Dachfläche möglichst groß wird.
(Die Module können horizontal oder vertikal installiert werden)



Beispiele für horizontale und vertikale Installation der Solarmodule

Aufgaben

Wir gehen davon aus, dass die Dachfläche rechteckig ist und keine Besonderheiten (Fenster, Gauben usw.) aufweist.
Der Benutzer gibt die Maße des Daches ein und erhält als Antwort den geeigneten Modultyp sowie die Anzahl der Module und die maximal belegbare Dachfläche. Es ist zu berücksichtigen, dass am Rand des Daches immer 10 cm frei bleiben müssen.

Es stehen folgende Module zur Auswahl:

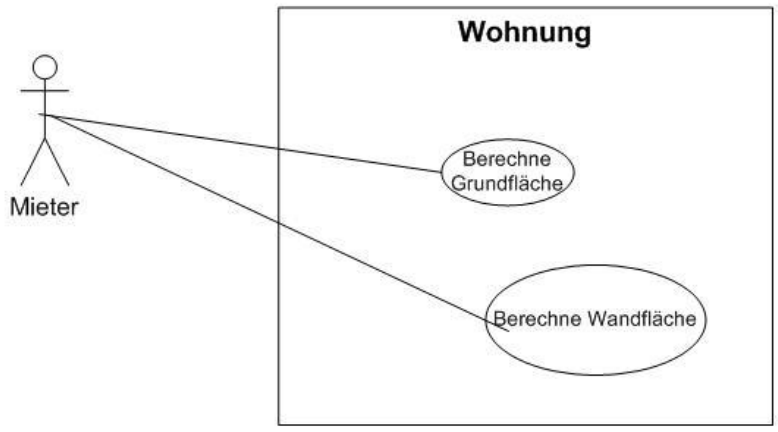
1. Alfasolar	1475 mm x 986 mm,
2. Sunpower	1560 mm x 800 mm
3. Conenergy	1490 mm x 1000 mm
4. Mitsubishi	1250 mm x 800 mm
5. Kyocera	1225 mm x 1225 mm
6. Solarworld	1675 mm x 1000 mm

1. Erstellen Sie eine Klasse **Modul** und schreiben Sie eine Applikation, die diese Aufgabe löst.
2. Peter Cornelius möchte die Software als **Java-Applet** auf der Homepage der Firma Sunpower installieren, damit der Kunde die geeigneten Module schon vorher auswählen kann. Lösen Sie das Problem.

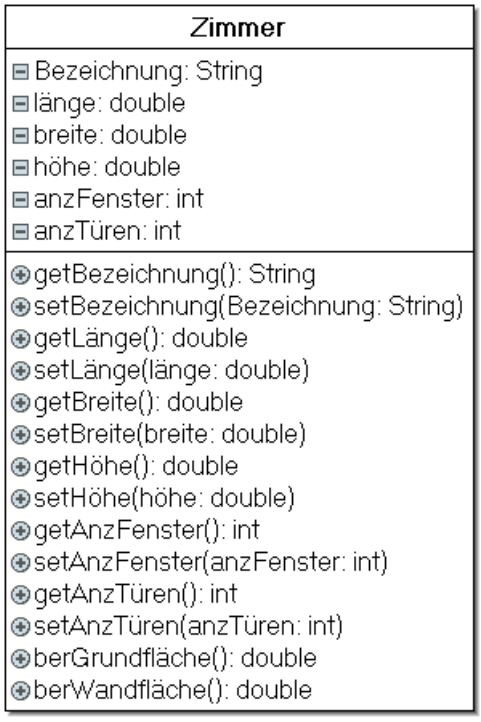
Lösungsteil

Fallbeispiel 1 - Renovieren:

zu a) Anwendungsfalldiagramm



zu b) Klassendiagramm



zu c)

```
public class Zimmer {

 // Anfang Variablen
 private String Bezeichnung;
 private double länge;
 private double breite;
 private double höhe;
 private int anzFenster;
 private int anzTüren;
 // Ende Variablen

 // Anfang Ereignisprozeduren

 public double getLänge() {
  return länge;
 }

 public void setLänge(double länge) {
  this.länge = länge;
 }

 public double berGrundfläche()
 {
   return this.breite*this.länge;
 }
 public double berWandfläche()
 {
  double w,w1,w2,tf,ff;
  w1=this.länge*this.höhe*2;
  w2=this.breite*this.höhe*2;
  tf=this.anzTüren*1*2;
  ff=this.anzFenster*1*0.8;
  w=w1+w2-(tf+ff);
  return w;
 }
 public String getBezeichnung() {
  return Bezeichnung;
 }

 public void setBezeichnung(String Bezeichnung)
{
  this.Bezeichnung = Bezeichnung;
 }

 // Ende Ereignisprozeduren
}
```

Zur Verkürzung wird hier lediglich die get- und set-Methode für das Attribut **länge** dargestellt.

Die Methode ***berGrundfläche*** soll die Grundfläche eines Zimmer berechnen. Sie benötigt dazu die beiden Attribute **länge** und **breite**. Die Methode enthält keine Übergabeparameter, da die Werte für die Länge und die Breite schon bei der Erzeugung des Objekts mit einer set-Methode festgelegt wurden. Das Ergebnis der Berechnung **länge * breite** wird mit **return** zurückgeliefert.

Die Methode ***beWandfläche*** soll die Fläche aller Zimmerwände berechnen. Sie benötigt dazu die Attribute **länge, breite** und **höhe**. Außerdem sind die Flächen für die Türen und die Fenster abzuziehen.
Aus Vereinfachungsgründen wollen wir eine Tür mit den Maßen 1 m x 2 m und ein Fenster mit den Maßen 1 m * 0,8 m ansetzen.
Die Methode enthält ebenfalls keine Übergabeparameter.

```
public class wohnen
{
 public static void main(String argv[])
 {
  Zimmer z1= new Zimmer( );
  z1.setLänge(5);
  z1.setBreite(4);
  z1.setHöhe(2.4);
  z1.setAnzFenster(3);
  z1.setAnzTüren(2);

//wir rufen die Methoden zur Berechnung der Grundfläche und der Wandfläche auf und lassen den
//Rückgabewert am Bildschirm anzeigen

  System.out.println("Grundfläche: "+z1.berGrundfläche());
  System.out.println("Wandfläche: "+z1.berWandfläche());
 }
}
```

Der Quellcode links erzeugt ein Objekt mit der internen Bezeichnung z1 und initialisiert alle Attribute mit einer set-Methode.

zu d)

```
public Zimmer(String z,double l, double b, double h, int f, int t)
 {
  this.Bezeichnung=z;
  this.länge=l;
  this.breite=b;
  this.höhe=h;
  this.anzFenster=f;
  this.anzTüren=t;
 }
```

```
public class wohnen
{
 public static void main(String argv[])
 {
  Zimmer z1= new Zimmer("Wohnzimmer",5,4,2.40,3,2);
  Zimmer z2= new Zimmer("Schlafzimmer",4,4,2.40,2,1);
  Zimmer z3= new Zimmer("Küche",3,3,2.40,1,1);

  System.out.println(""+z1.getBezeichnung()+" Grundfläche: "+z1.berGrundfläche());
  System.out.println(""+z1.getBezeichnung()+" Wandfläche: "+z1.berWandfläche());
  System.out.println(""+z2.getBezeichnung()+" Grundfläche: "+z2.berGrundfläche());
  System.out.println(""+z2.getBezeichnung()+" Wandfläche: "+z2.berWandfläche());
  System.out.println(""+z3.getBezeichnung()+" Grundfläche: "+z3.berGrundfläche());
  System.out.println(""+z3.getBezeichnung()+" Wandfläche: "+z3.berWandfläche());
 }
}
```

zu e)

Alle Objekte der Klasse Zimmer sollen in einem Array gespeichert werden.

1. Wir erzeugen ein Array vom Datentyp der Klasse, die wir verarbeiten wollen.

```
Zimmer [] z = new Zimmer[5];
```
Dieses Array kann 5 Zimmerobjekte aufnehmen.

2. Wir erzeugen Objekte und speichern diese im Array

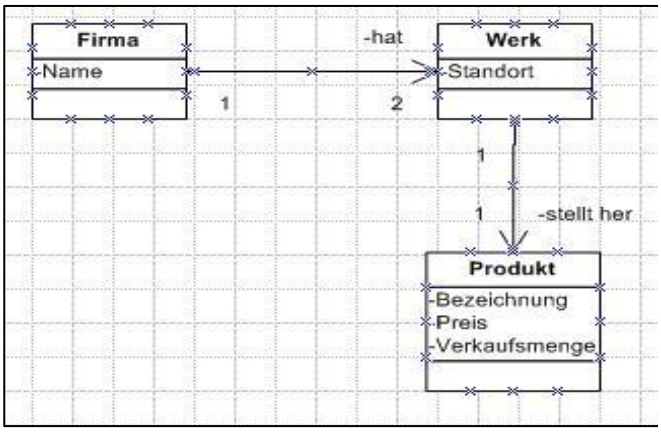
```
z[0] = new Zimmer("Wohnzimmer",5,4,2.40,3,2);
z[1] = new Zimmer("Schlafzimmer",4,4,2.40,2,1);   usw.
```

3. Die Verarbeitung, Auswertung usw. geschieht über eine Schleife.

```
double summe=0
for(int i=0; i<z.length; i++)
{
  summe=summe+z[i].berGrundfläche() );
}
```

Fallbeispiel 2 - Firmenumsatz - Opal:

zu 1. Klassendiagramm



zu 2-7. Fertige Klassen (endgültige Fassung)

```
public class Firma {
  // Attribute
  private String Name;
  private double gUmsatz;
//Aufgabenteil 7
  // Ende Attribute

  // Anfang Ereignisprozeduren

  public String getName() {
    return Name;
  }

  public void setName(String
Name) {
    this.Name = Name;
  }
  public double getGUmsatz()
  {
    return this.gUmsatz;
  }
  public void
BerechneFirmenumsatz(double
wu)
  {

this.gUmsatz=this.gUmsatz+wu;
  }
```

```
public class Werk {
  // Attribute
  private String Standort;
  private Produkt
meinProdukt;
//Aufgabenteil 5
  private double
WerksUmsatz;
//Aufgabenteil 6

  // Methoden
  public void
setStandort(String n)
  {
    this.Standort=n;
  }
  public String getStandort()
  {
    return this.Standort;
  }
  public void
setMeinProdukt(Produkt p)
  {
    this.meinProdukt=p;
  }
  public Produkt
getMeinProdukt()
  {
    return meinProdukt;
  }
  public double
bWerksUmsatz()
  {

this.WerksUmsatz=this.getM
einProdukt().getPreis()*this.g
etMeinProdukt().getVerkaufs
menge();
```

```
public class Produkt {
  // Attribute

  String Bezeichnung;
  float Preis;
  int Verkaufsmenge;

  // Methoden

  public void setBezeichnung(String
n)
  {
    this.Bezeichnung=n;
  }
  public String getBezeichnung()
  {
    return this.Bezeichnung;
  }
  public void setPreis(float p)
  {
    this.Preis=p;
  }
  public float getPPreis()
  {
    return this.Preis;
  }
  public void setVerkaufsmenge(int a)
  {
    this.Verkaufsmenge=a;
  }
  public int getVerkaufsmenge()
  {
    return this.Verkaufsmenge;

  }
```

```
public class Umsatz5
{
  public static void main(String[] args)
  {
    //Aufgabenteil 3
    Firma f = new Firma();
    f.setName("Opal");
    Werk w1 = new Werk();
    Werk w2 = new Werk();
    w1.setStandort("Dortmund");
    w2.setStandort("Essen");
    Produkt p1 = new Produkt();
    Produkt p2=new Produkt();
    p1.setBezeichnung("Smaragd");
    p1.setPreis(20000);
    p1.setVerkaufsmenge(150);
    p2.setBezeichnung("Saphir");
    p2.setPreis(16500);
    p2.setVerkaufsmenge(220);
    //Neues Werk in Siegen   Aufgabenteil 4
    Werk w3 = new Werk();
    w3.setStandort("Siegen");
    LKW l1 = new LKW();
    l1.setBezeichnung("Mammut");
    l1.setPreis(80000);
    l1.setVerkaufsmenge(30);
    l1.setZuladung(18000);
    //Aufgabenteil 5
    w1.setMeinProdukt(p1);
    w2.setMeinProdukt(p2);
    w3.setMeinProdukt(l1);
    //Aufgabenteil 6
    System.out.println("Werk: "+w1.getStandort()+" Produkt:
"+w1.getMeinProdukt().getBezeichnung()+", Umsatz: "+w1.bWerksUmsatz());
    System.out.println("Werk: "+w2.getStandort()+" Produkt:
"+w2.getMeinProdukt().getBezeichnung()+", Umsatz: "+w2.bWerksUmsatz());
    System.out.println("Werk: "+w3.getStandort()+" Produkt:
"+w3.getMeinProdukt().getBezeichnung()+", Umsatz: "+w3.bWerksUmsatz());
    //Aufabenteil 7
    f.BerechneFirmenumsatz(w1.bWerksUmsatz());
    f.BerechneFirmenumsatz(w2.bWerksUmsatz());
    f.BerechneFirmenumsatz(w3.bWerksUmsatz());
    System.out.println("Gesamtumsatz der Firma "+f.getName()+" "+f.getGUmsatz());
    }
}
```

```
public class LKW extends Produkt {

  // Anfang Variablen
  private int Zuladung;
  // Ende Variablen

  // Anfang Ereignisprozeduren
  public int getZuladung() {
    return Zuladung;
  }

  public void setZuladung(int Zuladung) {
    this.Zuladung = Zuladung;
  }

  // Ende Ereignisprozeduren
}
```

Fallbeispiel 3 – Kunde und Betreuer:

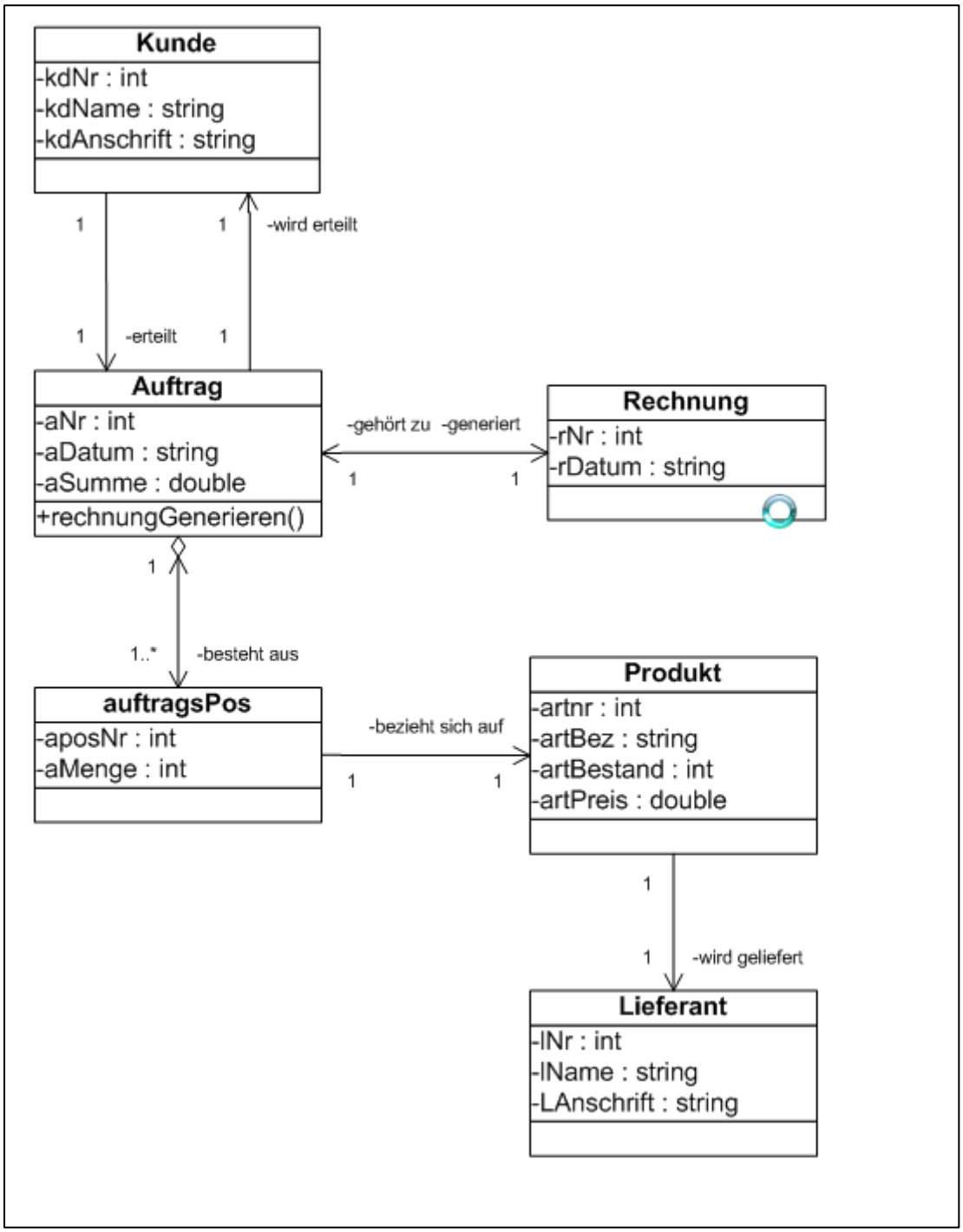
Lösung für alle drei Anwendungsfälle

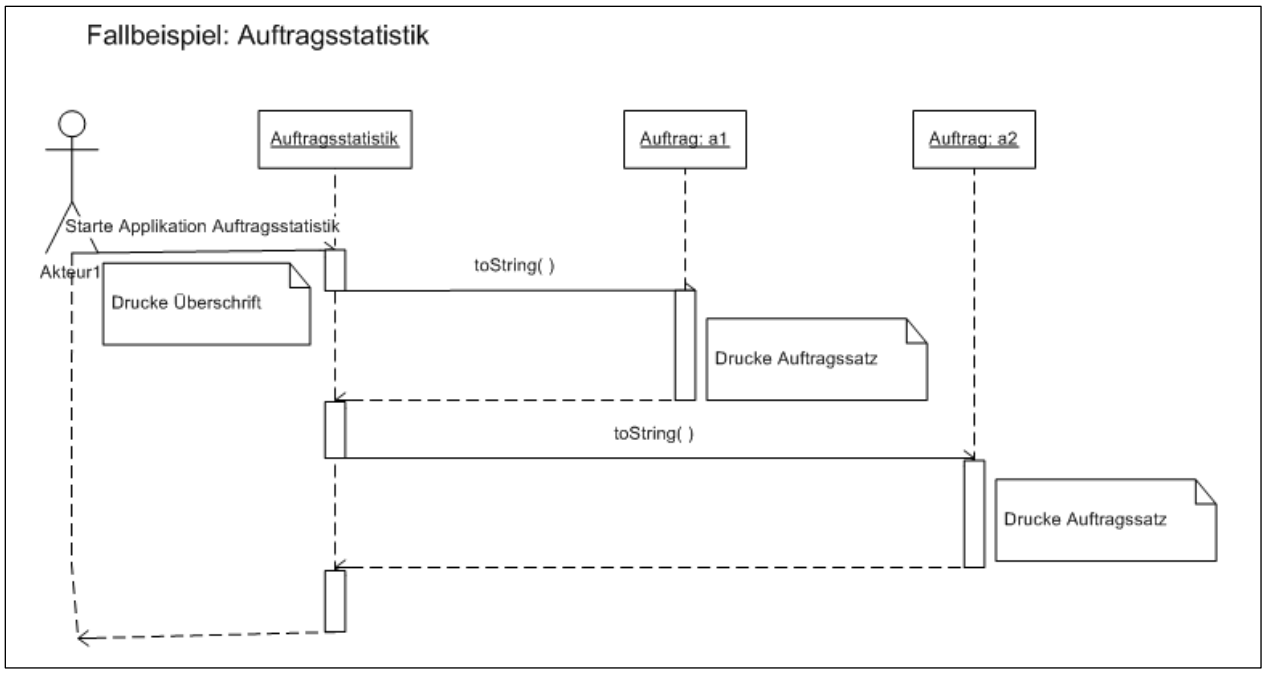
```
import java.io.*;
public class Kundenbetreuung2
{
  public static void main(String[] args) throws IOException
  {
    int wahl=9;
    //Arrays für Klassenobjekte erzeugen
    Kunde[] k = new Kunde[5];
    Betreuer [] b = new Betreuer[5];
    //Objekte erzeugen und intialiseren
    k[0]= new Kunde(101,"Meier");
    b[0]= new Betreuer(1001,"Hans");
    k[0].setKBetreuer(b[0]);
    k[1]= new Kunde(102,"Wagner");
    b[1]= new Betreuer(1002,"Alma");
    k[1].setKBetreuer(b[1]);
    k[2]= new Kunde(103,"Klee & Munk");
    b[2]= new Betreuer(1003,"Fritz");
    k[2].setKBetreuer(b[2]);
    k[3]= new Kunde(104,"Koch GmbH");
    b[3]= new Betreuer(1004,"Ali");
    k[3].setKBetreuer(b[3]);
    k[4]= new Kunde(105,"Weber OHG");
    b[4]= new Betreuer(1005,"Hassan");
    k[4].setKBetreuer(b[4]);

    do
    {
       System.out.println("\nAnwendungsfälle");
       System.out.println("\n1 - Kunde erteilt Auftrag");
       System.out.println("2 - Kunde bezahlt");
       System.out.println("3 - Provision anzeigen");
       System.out.println("0 - Beenden");
       System.out.print("\n\nEingabe ");
       wahl=Console.i();
       switch(wahl)
       {
            case 1:    auftrag(k,b); break;
            case 2:    bezahlen(k,b); break;
            case 3:    provision(b);  break;
            case 0:    break;
            default:   System.out.println("\nUngültige Eingabe");
       }
    }
    while(wahl>0);

  }
  public static void auftrag(Kunde [] k,Betreuer [] b) throws IOException
  {
        int kd;
        int merk=-1;
```

```
        double s;
        System.out.print("\nKundennummer: ");
        kd=Console.i();
        for (int i=0;i<5; i++)
        {
          if (k[i].getKdnr()==kd)
          {
            merk=i;
          }
        }
        if(merk>=0)
        {
           System.out.print("Auftragssumme: ");
           s=Console.d();
           k[merk].setKUmsatz(s);
           k[merk].setKOffen(s);
           System.out.println("\nKunde "+k[merk].getKName()+" hat noch "+k[merk].getKOffen()+" zu
bezahlen");
        }
        else
        {
          System.out.println("Kunde nicht gefunden");
        }
  }
  public static void bezahlen(Kunde [] k,Betreuer [] b) throws IOException
  {
        int kd;
        int merk=-1;
        double s;
        System.out.print("\nKundennummer: ");
        kd=Console.i();
        for (int i=0;i<5; i++)
        {
          if (k[i].getKdnr()==kd)
          {
            merk=i;
          }
        }
        if(merk>=0)
        {
           System.out.print("Bezahlter Betrag: ");
           s=Console.d();
           k[merk].kundeBezahlt(s);
        }
        else
        {
          System.out.println("Kunde nicht gefunden");
        }
    }

  public static void provision(Betreuer [] b)  throws IOException
  {
        int bd;
        int merk=-1;
        double s;
        System.out.print("\nBetreuer: ");
        bd=Console.i();
```

```
        for (int i=0;i<5; i++)
        {
          if (b[i].getBnr()==bd)
          {
            merk=i;
          }
        }
        System.out.println("Provision: "+b[merk].getBProvision());
   }
}
```

Fallbeispiel 4 - Auftragsstatistik:

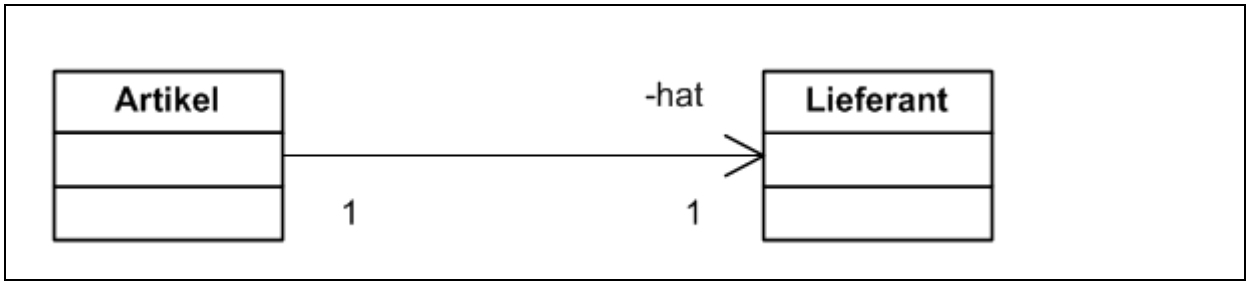




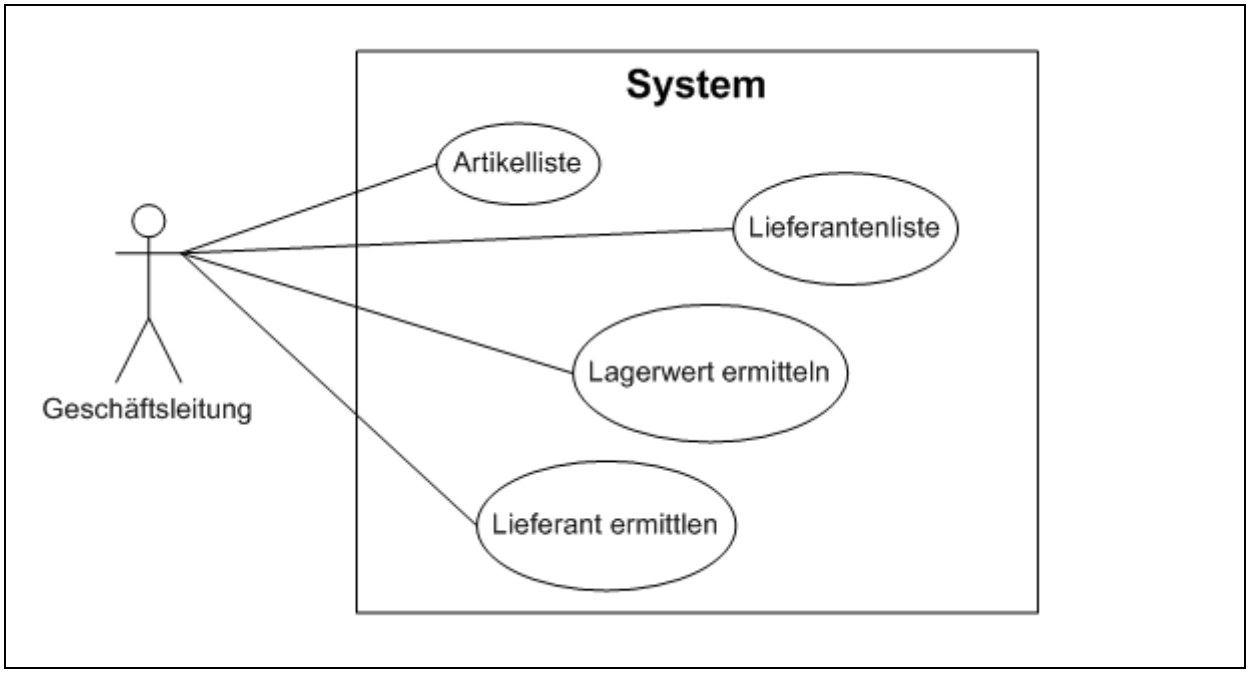
Im Sequenzdiagramm wird davon ausgegangen, dass es eine toString-Methode gibt, die die Werte des Auftrags in der Form, wie in der Aufgabe beschrieben, ausgibt.

Fallbeispiel 5 - Wer liefert was? :

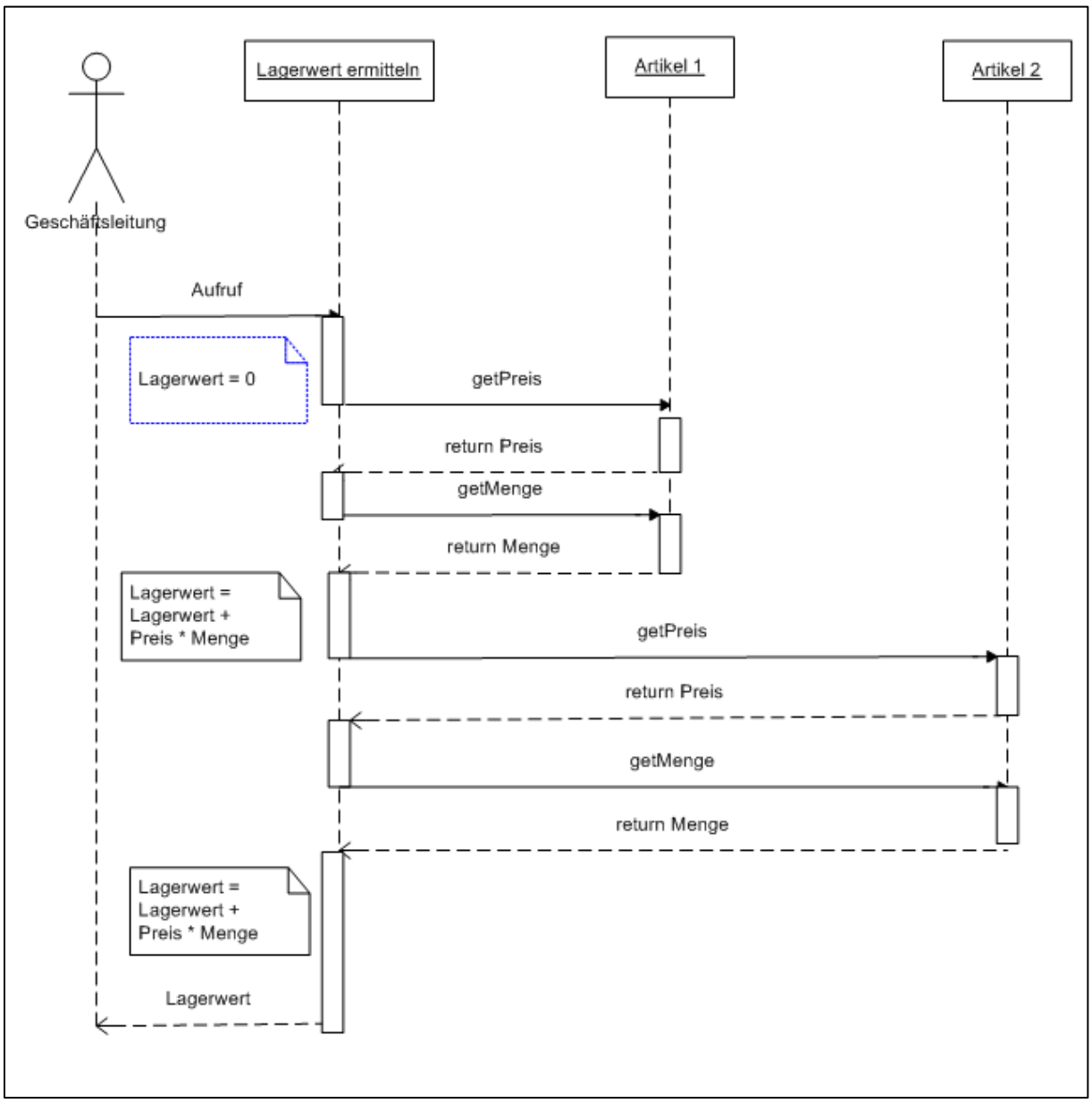
1. Klassendiagramm (ohne Attribute und Methoden)



2. Anwendungsfalldiagramm (Use-Case-Diagramm)



3. Sequenzdiagramm (eine von mehreren Lösungsmöglichkeiten)



Quellcode

1. Klassen

```
public class Artikel {

  // Anfang Variablen
  private int Artnr;
  private String Artbez;
  private int ABestand;
  private double AEkPreis;
  private Lieferant meinLieferant;
  // Ende Variablen

  // Anfang Ereignisprozeduren
  public Artikel(int a, String b, int m, double p)
  {
    this.Artnr=a;
    this.Artbez=b;
    this.ABestand=m;
    this.AEkPreis=p;
  }
  public int getArtnr() {
    return Artnr;
  }

  public void setArtnr(int Artnr) {
    this.Artnr = Artnr;
  }

  public String getArtbez() {
    return Artbez;
  }

  public void setArtbez(String Artbez) {
    this.Artbez = Artbez;
  }

  public int getABestand() {
    return ABestand;
  }

  public void setABestand(int ABestand) {
    this.ABestand = ABestand;
  }

  public double getAEkPreis() {
    return AEkPreis;
  }

  public void setAEkPreis(double AEkPreis) {
    this.AEkPreis = AEkPreis;
  }

  public Lieferant getMeinLieferant() {
    return meinLieferant;
  }

  public void setMeinLieferant(Lieferant meinLieferant) {
    this.meinLieferant = meinLieferant;
  }

  // Ende Ereignisprozeduren
}
```

```
public class Lieferant {

  // Anfang Variablen
  private int Lnr;
  private String LName;
  private String LAnschrift;
  // Ende Variablen

  // Anfang Ereignisprozeduren

  public Lieferant(int l, String n, String a)
  {
    this.Lnr=l;
    this.LName=n;
    this.LAnschrift=a;
  }
    public int getLnr() {
    return Lnr;
  }

  public void setLnr(int Lnr) {
    this.Lnr = Lnr;
  }

  public String getLName() {
    return LName;
  }

  public void setLName(String LName) {
    this.LName = LName;
  }

  public String getLAnschrift() {
    return LAnschrift;
  }

  public void setLAnschrift(String LAnschrift) {
    this.LAnschrift = LAnschrift;

  }
  // Ende Ereignisprozeduren
}
```

2. Applikationen

```
import java.io.*;

public class Artikelliste
{
  public static void main(String argv[]) throws IOException
  {
    int artikelnr;
    Artikel [] a = new Artikel[5];
    a[0] = new Artikel(100,"Hose",100,49.5);
    a[1] = new Artikel(101,"Hemd",85,27.5);
    a[2]= new Artikel(102,"Schuh",42,99.50);
    a[3]= new Artikel(103,"Strumpf",50,9.50);
    a[4]= new Artikel(104,"Jacke",30,75.00);
    Lieferant [] l = new Lieferant[3];
    l[0] = new Lieferant(200,"Huber AG","Frankfurt");
    l[1] = new Lieferant(201,"Meier","Gießen");
    l[2] = new Lieferant(202,"Allkauf GmbH","Hanau");
    a[0].setMeinLieferant(l[0]);
    a[1].setMeinLieferant(l[1]);
    a[2].setMeinLieferant(l[2]);
    a[3].setMeinLieferant(l[2]);
    a[4].setMeinLieferant(l[2]);
    System.out.println("Artikelliste\n");
```

```
    for(int i=0;i<5;i++)
    {

System.out.println(a[i].getArtnr()+"\t"+a[i].getArtbez()+"\t"+a[i].getABestand()+"\t"+a[i].get
AEkPreis());
    }
  }
```

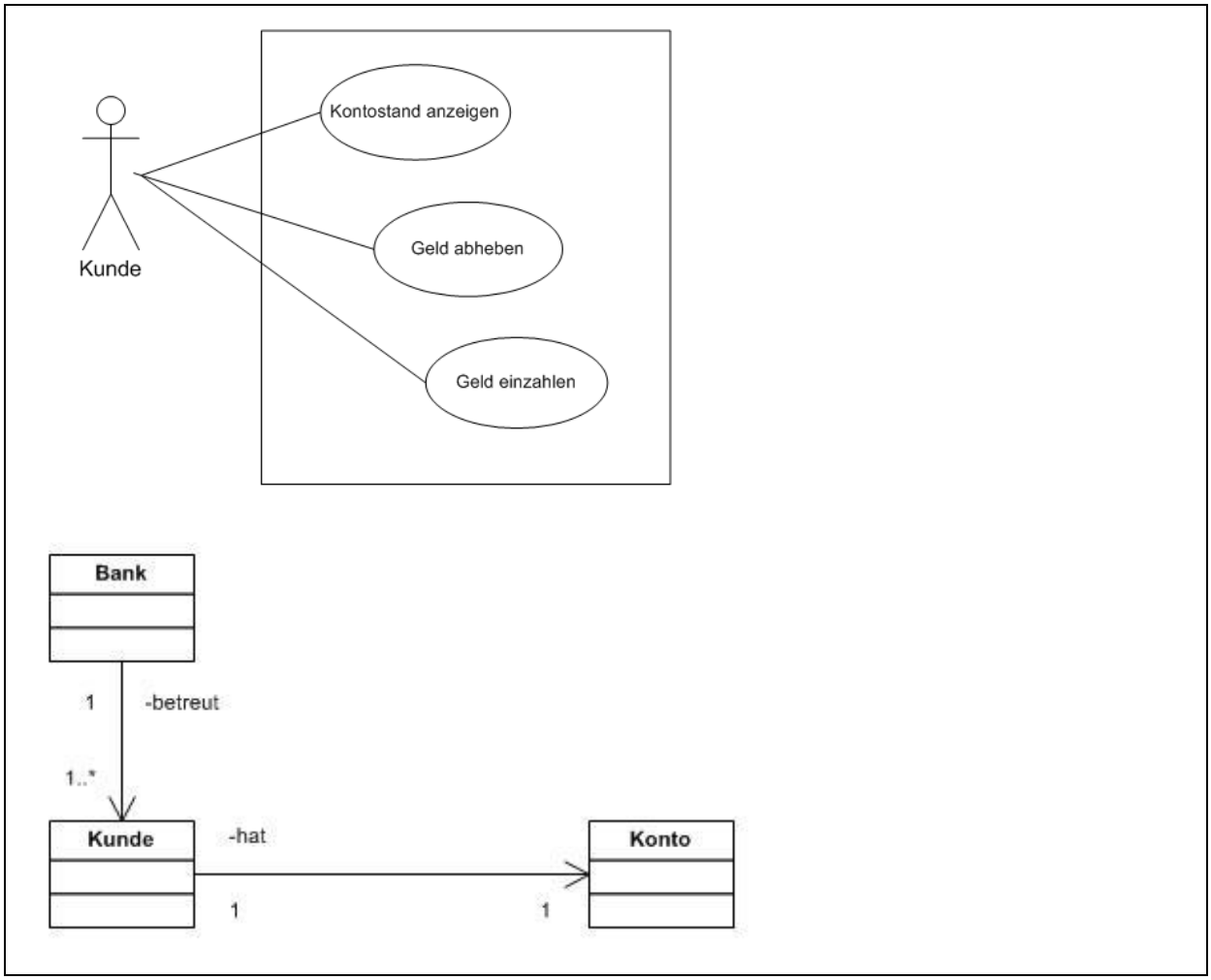
```
import java.io.*;

public class Lieferantenliste
{
  public static void main(String argv[]) throws IOException
  {
    int artikelnr;
    Artikel [] a = new Artikel[5];
    a[0] = new Artikel(100,"Hose",100,49.5);
    a[1] = new Artikel(101,"Hemd",85,27.5);
    a[2]= new Artikel(102,"Schuh",42,99.50);
    a[3]= new Artikel(103,"Strumpf",50,9.50);
    a[4]= new Artikel(104,"Jacke",30,75.00);
    Lieferant [] l = new Lieferant[3];
    l[0] = new Lieferant(200,"Huber AG","Frankfurt");
    l[1] = new Lieferant(201,"Meier","Gießen");
    l[2] = new Lieferant(202,"Allkauf GmbH","Hanau");
    a[0].setMeinLieferant(l[0]);
    a[1].setMeinLieferant(l[1]);
    a[2].setMeinLieferant(l[2]);
    a[3].setMeinLieferant(l[2]);
    a[4].setMeinLieferant(l[2]);

    System.out.println("Lieferantenliste\n");
    for(int i=0;i<3;i++)
    {
      System.out.println(l[i].getLnr()+"\t"+l[i].getLName()+"\t"+l[i].getLAnschrift());
    }
  }
}
```

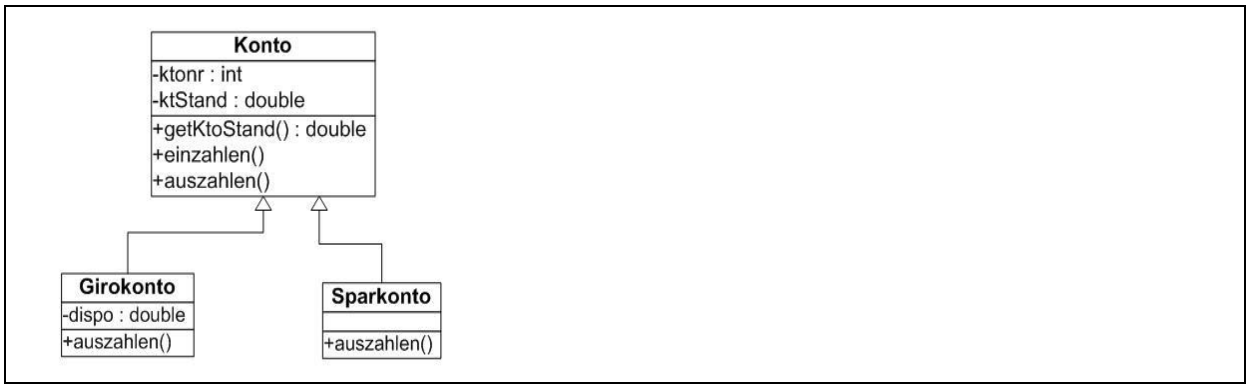
```
import java.io.*;

public class Lagerwert
{
  public static void main(String argv[]) throws IOException
  {
    int artikelnr;
    Artikel [] a = new Artikel[5];
    a[0] = new Artikel(100,"Hose",100,49.5);
    a[1] = new Artikel(101,"Hemd",85,27.5);
    a[2]= new Artikel(102,"Schuh",42,99.50);
    a[3]= new Artikel(103,"Strumpf",50,9.50);
    a[4]= new Artikel(104,"Jacke",30,75.00);

    double lw=0;
    System.out.println("Lagerwert\n");
    for(int i=0;i<5;i++)
    {
       lw=lw+a[i].getABestand()*a[i].getAEkPreis();
    }
    System.out.println("Gesamtwert aller Artikel zum EK: "+lw);
  }
}
```

```
import java.io.*;

public class werLiefertWas
{
  public static void main(String argv[]) throws IOException
  {
    int artikelnr;
    Artikel [] a = new Artikel[3];
    a[0] = new Artikel(100,"Hose",100,49.5);
    a[1] = new Artikel(101,"Hemd",85,27.5);
    a[2]= new Artikel(102,"Schuh",42,99.50);
    Lieferant l1=new Lieferant(200,"Huber AG","Frankfurt");
    Lieferant l2=new Lieferant(201,"Meier","Gießen");
    Lieferant l3=new Lieferant(202,"Allkauf GmbH","Hanau");
    a[0].setMeinLieferant(l1);
    a[1].setMeinLieferant(l2);
    a[2].setMeinLieferant(l3);
    System.out.print("Artikelnummer eingeben: ");
    artikelnr=Console.i();                          //siehe Anmerkung
    int z,merk=0;
    boolean g=false;
    for(z=0; z<3; z++)
    {
      if(artikelnr==a[z].getArtnr())
      {
         g=true;
         merk=z;
      }
    }
    if(g)
      System.out.println("Artikel "+a[merk].getArtnr()+" "+a[merk].getArtbez()+", Lieferant:
"+a[merk].getMeinLieferant().getLName());
    else
      System.out.println("Artikel nicht gefunden!");
  }
}
```

Anmerkung: Die Tastatureingabe wird über eine die Klasse Console realisert, die sich in diesem Fall im gleichen Verzeichnis befinden muss, wie die Applikation. Hier wird die Methode ***i( )*** der Klasse **Console** aufgerufen, die einen Integerwert einliest.

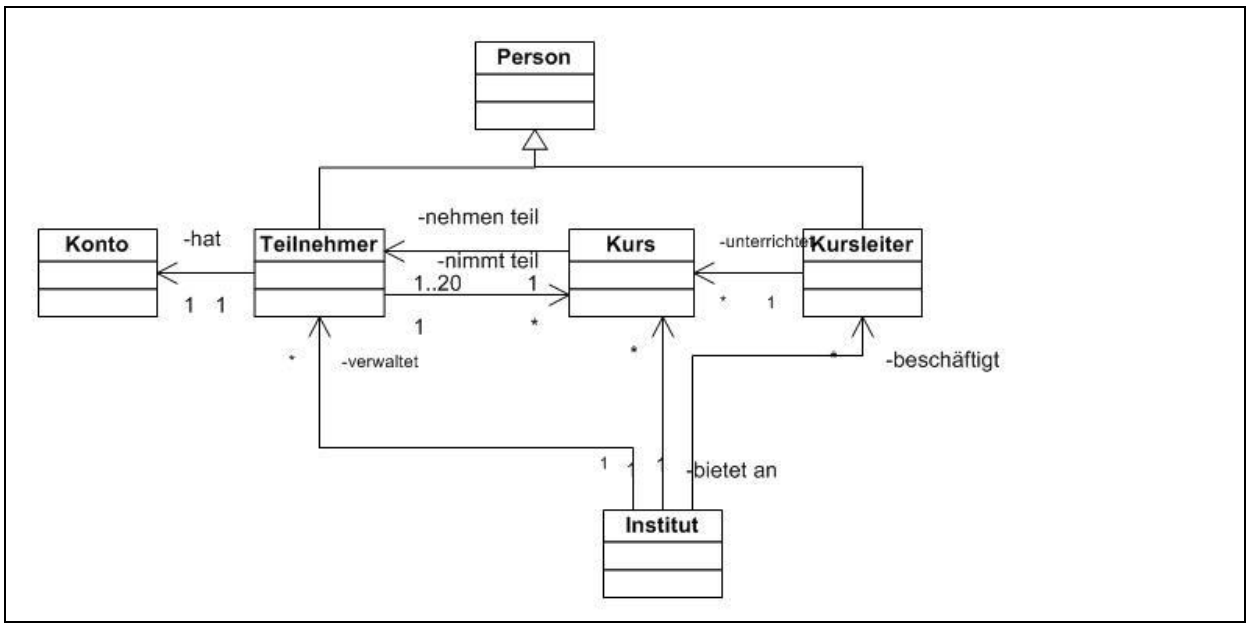
Fallbeispiel 6 - Geldautomat:

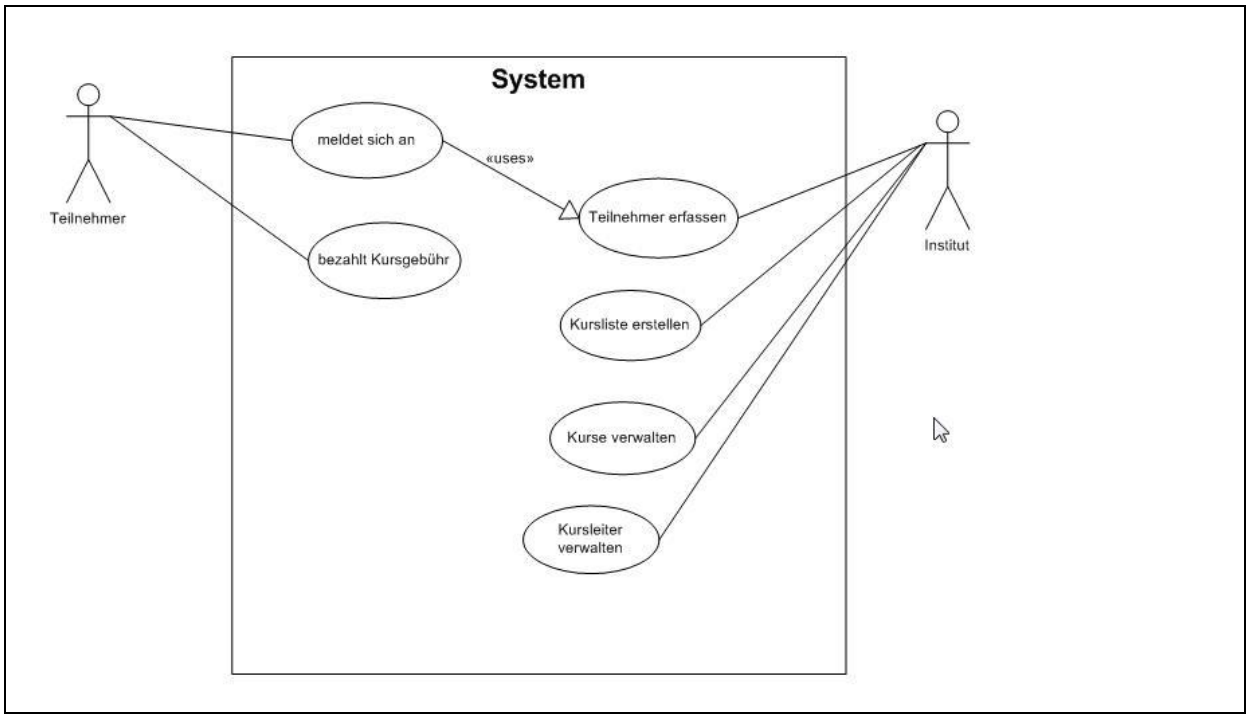




Da der Quellcode doch recht umfangreich ist, wird hier auf die Darstellung einer möglichen Lösung verzichtet und lediglich eine vereinfachte UML-Darstellung gezeigt.

Fallbeispiel 7 - Kursverwaltung:





Weitere Anwendungsfälle sind denkbar.

c) Die Erstellung der Kursliste kann durch die Assoziationen zwischen Teilnehmer und Kurs sowie zwischen Kurs und Institut realisiert werden. Die bidirektionale Assoziation zwischen Teilnehmer und Kurs ermöglicht der Klasse **Kurs** ein Verweisattribut in Form eines Containers (z. B. Array) in dem alle Instanzen erfasst werden, die sich für einen Kurs angemeldet haben. Im Anwendungsfall ‚Kursliste erstellen' kann das Institut über die Assoziation zu einem Kurs über ein Wiederholungskonstrukt alle Elemente des Containers wieder auslesen und die gewünschten Attribute (z. B. über eine toString-Methode) ausgeben lassen.

Fallbeispiel 8 – Wer sitzt wo? :

```
import java.io.*;
public class Platzreservierung
{
  public static int n=30;
  public static int b_index=0;
  public static void main(String argv[]) throws IOException
  {
    Platz [] pArray= new Platz[n];
    Besucher [] bArray = new Besucher[n];
    int ausw;
    objekte(pArray);     //es werden  n Objekte für Plätze erzeugt
    do
    {
      System.out.println("\n\nMenü\n");
      System.out.println("\n1 - Platzliste anzeigen");
      System.out.println("\n2 - Freie Plätze suchen");
      System.out.println("\n3 - Platz buchen");
      System.out.println("\n4 - Belegte Plätze anzeigen");
      System.out.println("\n0 - System beenden");
      System.out.print("\n\nEingabe: ");
      ausw=Console_IO.I_int();
      switch(ausw)
      {
                case 1:  anzeigen(pArray);break;
                case 2:  freiAnzeigen(pArray);break;
                case 3:  buchen(pArray,bArray);break;
                case 4:  belegungAnzeigen(pArray,bArray);break;
      }
    }
    while(ausw!=0);

   }
    public static void objekte(Platz pArray[])
    {
       int i=0;
       for(int b=1; b<=3;b++)
       {
          for(int r=1; r<=2;r++)
          {
           for(int p=1; p<=5; p++)
           {
                  pArray[i]=new Platz(b,r,p);
                  i++;
           }
          }
       }
    }

    public static void anzeigen(Platz pArray[])
    {
       System.out.println("Platzliste\n");
       System.out.println("Platznr Block  Reihe  Platz  belegt");
       for(int k=0; k<n;k++)
       {
        System.out.println(+(k+1)+"         "+pArray[k].getBlock()+"      "+pArray[k].getReihe()+"
"+pArray[k].getNummer()+"    "+pArray[k].getBelegt());
       }
    }
```

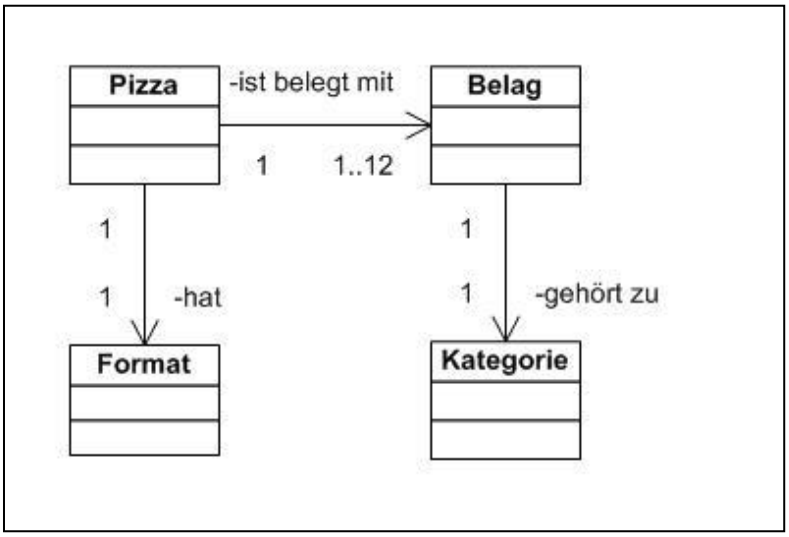
```
   public static void freiAnzeigen(Platz pArray[])
   {
      System.out.println("Platzliste\n");
      System.out.println("Platznr Block  Reihe  Platz  belegt");
      for(int k=0; k<n;k++)
      {
       if(pArray[k].getBelegt()==false)
       {
          System.out.println(+(k+1)+"       "+pArray[k].getBlock()+"      "+pArray[k].getReihe()+"
"+pArray[k].getNummer()+"    "+pArray[k].getBelegt());
       }

      }
   }

   public static void buchen(Platz pArray[],Besucher bArray[])  throws IOException
   {
      int pwahl;
      String name;
      System.out.println("Platzliste\n");
      System.out.println("Platznr Block  Reihe  Platz  belegt");
      for(int k=0; k<n;k++)
      {

       if(pArray[k].getBelegt()==false)

       {
          System.out.println(+(k+1)+"         "+pArray[k].getBlock()+"      "+pArray[k].getReihe()+"
"+pArray[k].getNummer()+"    "+pArray[k].getBelegt());
       }
      }
      System.out.print("\nAuswahl einer Platznummer: ");
      pwahl=Console_IO.I_int();
      pArray[pwahl-1].setBelegt(true);
      System.out.print("\nName: ");
      name=Console_IO.I_String();
      bArray[b_index]=new Besucher(name);
      pArray[pwahl-1].setBucher(bArray[b_index]);
      b_index++;
      System.out.println("\n Platznummer "+pwahl+" ist gebucht von: "+name);
   }

   public static void belegungAnzeigen(Platz pArray[], Besucher bArray[])

   {

      System.out.println("Belegte Plätze\n");

      System.out.println("Platznr Block  Reihe  Platz  belegt von");

      for(int k=0; k<n;k++)

      {

       if(pArray[k].getBelegt()==true)

       {

          System.out.println("   "+(k+1)+"       "+pArray[k].getBlock()+"      "+pArray[k].getReihe()+"
"+pArray[k].getNummer()+"    "+pArray[k].getBucher().getName());

       }

      }
   }}
```

```
public class Platz {

 // Anfang Attribute
 // Anfang Variablen
 private int Nummer;
 private int Reihe;
 private boolean belegt;
 private int Block;
 private Besucher bucher;
 // Ende Variablen
 // Ende Attribute

 // Anfang Methoden
 public Platz(int b, int r, int n)
 {
   this.Block=b;
   this.Reihe=r;
   this.Nummer=n;
   this.belegt=false;

 }
 // Anfang Ereignisprozeduren
 public int getReihe() {
   return Reihe;
 }

 public void setReihe(int Reihe) {
   this.Reihe = Reihe;
 }

 public int getNummer() {
   return Nummer;
 }

 public void setNummer(int Nummer) {
   this.Nummer = Nummer;
 }

 public boolean getBelegt() {

   return this.belegt;
 }

 public void setBelegt(boolean belegt) {
   this.belegt = belegt;
 }

 public int getBlock() {
   return Block;
 }

 public void setBlock(int Block) {
   this.Block = Block;
 }
 public Besucher getBucher() {
   return bucher;
 }

 public void setBucher(Besucher bucher) {
   this.bucher = bucher;
```

```
  }

  // Ende Ereignisprozeduren

  // Ende Methoden
}
```

```
public class Besucher {

  // Anfang Attribute
  private String Name;
  // Ende Attribute

  // Anfang Methoden
  public Besucher(String n)
  {
    this.Name=n;
  }
  public String getName() {
    return Name;
  }

  public void setName(String Name) {
    this.Name = Name;
  }

  // Ende Methoden
}
```

Fallbeispiel 9 - Pizzaservice:




```
import java.io.*;
public class pizzaservice
{
  public static void main(String argv[])  throws IOException
  {

    int gr, belag=99;
    int best,wahl;
    // Instanzen für Kategorien, Größe und Beläge anlegen
    Pizzaformat [] a_Pizzaformat = new Pizzaformat[2];
    a_Pizzaformat[0]=new Pizzaformat(1,4.0);
    a_Pizzaformat[1]=new Pizzaformat(2,4.5);
    kategorie [] a_Kategorie= new kategorie[3];
    a_Kategorie[0] = new kategorie(0.5,0.7);
    a_Kategorie[1] = new kategorie(1.0,1.3);
    a_Kategorie[2] = new kategorie(1.2,1.6);
    Belag [] a_Belag = new Belag [12];
    a_Belag[0]=new Belag("Paprika",a_Kategorie[0]);
    a_Belag[1]=new Belag("Peperoni",a_Kategorie[0]);
    a_Belag[2]=new Belag("Knoblauch",a_Kategorie[0]);
    a_Belag[3]=new Belag("Zwiebeln",a_Kategorie[0]);
    a_Belag[4]=new Belag("Oliven",a_Kategorie[0]);
    a_Belag[5]=new Belag("Salami",a_Kategorie[1]);
    a_Belag[6]=new Belag("Schinken",a_Kategorie[1]);
    a_Belag[7]=new Belag("Ananas",a_Kategorie[1]);
    a_Belag[8]=new Belag("extra Käse",a_Kategorie[1]);
    a_Belag[9]=new Belag("Gorgonzola",a_Kategorie[2]);
    a_Belag[10]=new Belag("Muscheln",a_Kategorie[2]);
    a_Belag[11]=new Belag("Shrimps",a_Kategorie[2]);
    // Ende
    do
    {
      System.out.println("Pizzeria Max\n\n");
      System.out.println("1 - Stammdatenverwaltung (noch nicht fertig)\n");
      System.out.println("2 - Pizzabestellung\n");
      System.out.println("0 - Beenden\n");
      wahl=Console_IO.IO_int("Auswahlziffer: ");

      if (wahl==1)
```

```
    {
      System.out.println("1 - Grundpreise");        //Prozedur fehlt noch
      System.out.println("2 - Kategoriepreise");    //Prozedur fehlt noch
      System.out.println("3 - Belagänderungen");    //Prozedur fehlt noch
      wahl=Console_IO.IO_int("Auswahlziffer: ");
    }

    if(wahl==2)
    {
      do
      {
        pizza p=new pizza();
        double gesamtpreis=0.;
        System.out.println("Pizzeria Bella Italia - Pizzabestellung\n");
        System.out.println("1 = Pizza normal (28 cm)");
        System.out.println("2 = Pizza groß   (32 cm)");
        System.out.print("\nEingabe: ");
        gr=Console_IO.I_int();
        p.setMeinF(a_Pizzaformat[gr-1]);
      do
      {
        System.out.println("\nBelagauswahl\n");
        System.out.println(" 1 = Paprika");
        System.out.println(" 2 = Peperoni");
        System.out.println(" 3 = Knoblauch");
        System.out.println(" 4 = Zwiebeln");
        System.out.println(" 5 = Oliven");
        System.out.println(" 6 = Salami");
        System.out.println(" 7 = Schinken");
        System.out.println(" 8 = Ananas");
        System.out.println(" 9 = extra Käse");
        System.out.println("10 = Gorgonzola");
        System.out.println("11 = Muscheln");
        System.out.println("12 = Shrimps");
        System.out.println(" 0 = kein weiterer Belag");
        System.out.print("\nBelagauswahl: ");
        belag=Console_IO.I_int();
        if (belag>0)
        {
          p.setMbelag(a_Belag[belag-1]);
          //Preis berechnen
          gesamtpreis=gesamtpreis+p.berPreis();
         }
       }
       while(belag!=0);
       gesamtpreis=gesamtpreis+p.getMeinF().getPreis();
       System.out.println("\nSie haben zu zahlen: "+gesamtpreis+" Euro");
       System.out.print("\n\nNoch eine Bestellung (1/2)? ");
       best=Console_IO.I_int();
     }
     while(best==1);
   }
    }
    while(wahl>0);
  }
}
```

```
public class pizza {

  // Anfang Attribute
  private Pizzaformat meinF;
  private Belag mbelag;
  // Ende Attribute

  // Anfang Methoden
  public void setMeinF(Pizzaformat p)
  {
    this.meinF=p;
  }
  public Pizzaformat getMeinF()
  {
    return this.meinF;
  }
  public double berPreis()
  {
    double preis=0;
    if (this.meinF.getGröße()==1)
    {
      preis=preis+this.mbelag.getMkategorie().getKp1();
    }
    else
    {
      preis=preis+this.mbelag.getMkategorie().getKp2();
    }
    return preis;
  }
  public Belag getMbelag() {
    return this.mbelag;
  }

  public void setMbelag(Belag mbelag) {
    this.mbelag = mbelag;
  }

  // Ende Methoden
}
```

```
public class kategorie {

  // Anfang Attribute
  private double kp1;
  private double kp2;
  // Ende Attribute

  // Anfang Methoden
  public kategorie(double k1,double k2)
  {
    this.kp1=k1;
    this.kp2=k2;
  }
  public double getKp1() {
    return this.kp1;
  }

  public void setKp1(double kp) {
    this.kp2 = kp;
  }
```

```
  public double getKp2() {
    return this.kp2;
  }

  public void setKp2(double kp) {
    this.kp2 = kp;
  }
    // Ende Methoden
}
```

```
public class Pizzaformat {

  // Anfang Attribute
  private int Größe;
  private double preis;
  // Ende Attribute

  public Pizzaformat(int g,double p)
  {
   this.Größe=g;
   this.preis=p;
  }

  // Anfang Methoden

  public void setPreis(double Preis) {
    this.preis = Preis;
  }

  public int getGröße() {
    return Größe;
  }

  public void setGröße(int Größe) {
    this.Größe = Größe;
  }

  public double getPreis() {
    return preis;
  }

  // Ende Methoden
}
```

```
public class Belag {

  // Anfang Attribute
  private String belagname;
  private kategorie mkategorie;
  // Ende Attribute

  // Anfang Methoden
  public Belag(String n, kategorie k)
  {
    this.belagname=n;
    this.mkategorie=k;
  }

  public String getBelagname() {
    return belagname;
  }
```

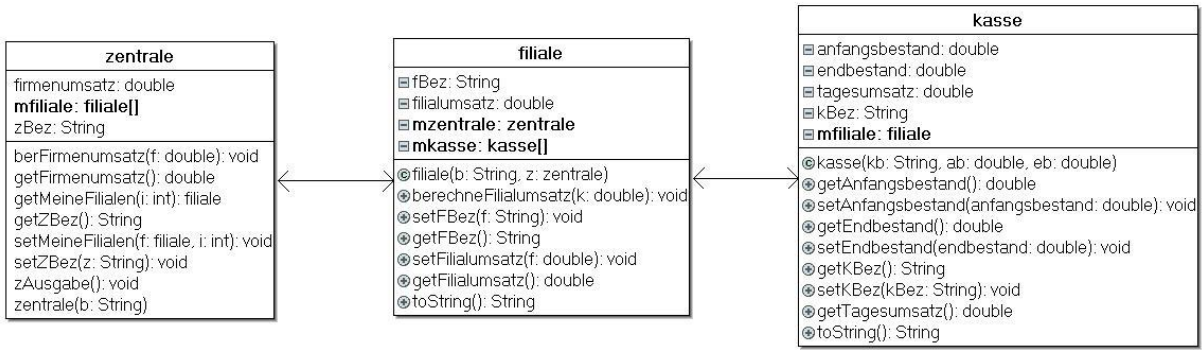
```
  public void setBelagname(String belagname) {
    this.belagname = belagname;
  }

  public kategorie getMkategorie() {
    return mkategorie;
  }

  public void setMkategorie(kategorie mkategorie) {
    this.mkategorie = mkategorie;
  }
  // Ende Methoden
}
```

Fallbeispiel 10 – Japan Wines:

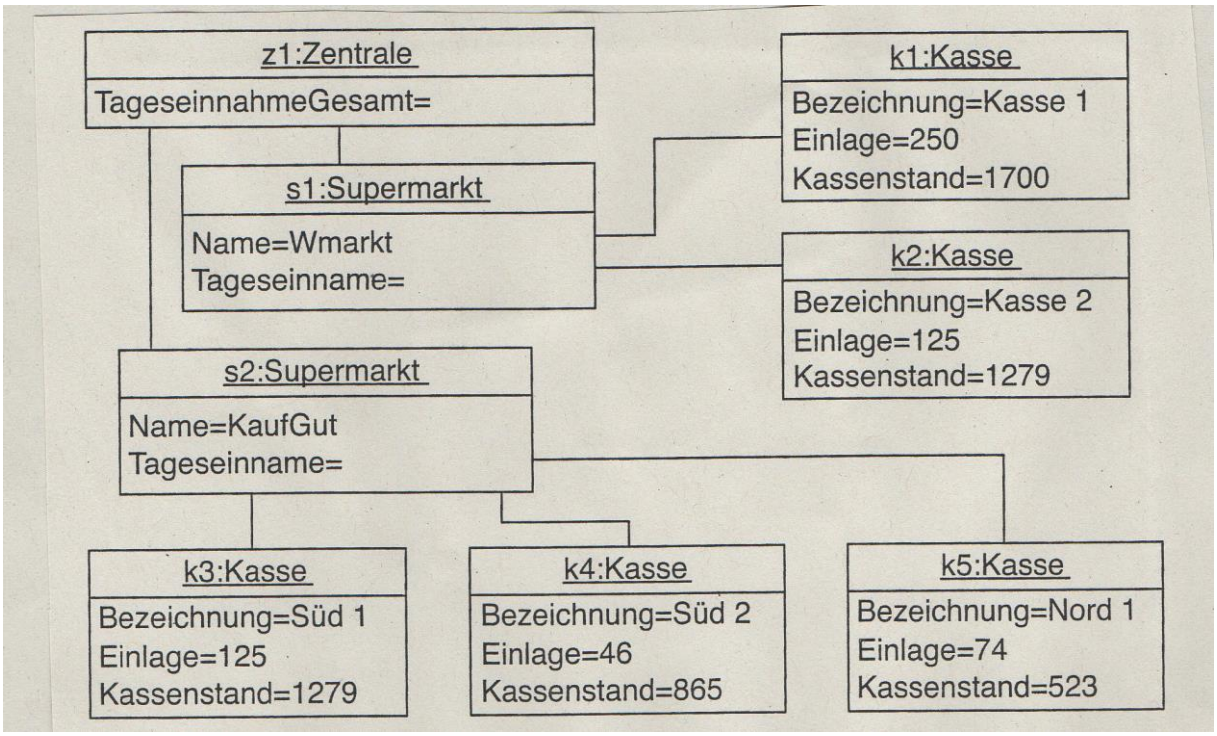
Das Projekt wurde in ähnlicher Form an der Universität Marburg durchgeführt. Auf die umfangreiche Darstellung von Lösungsvorschlägen wird hier verzichtet. Näheres findet sich unter den folgenden Links:

http://www.mathematik.uni-marburg.de/~hesse/uml/

Fallbeispiel 11 – Kassenumsätze:



In obigem Klassendiagramm wurden zwischen den Klassen jeweils bidirektionale Beziehungen verwendet. Zur Realisierung des Anwendungsfalles ‚Gesamtumsatz berechnen' genügen eindirektionale Beziehungen von der Zentrale zur Filiale und von der Filiale zur Kasse. Die Assoziation zwischen Filiale und Kasse kann auch als Aggregation dargestellt werden, da eine Kasse ohne den dazugehörigen Markt eher nicht selbständig sein kann.



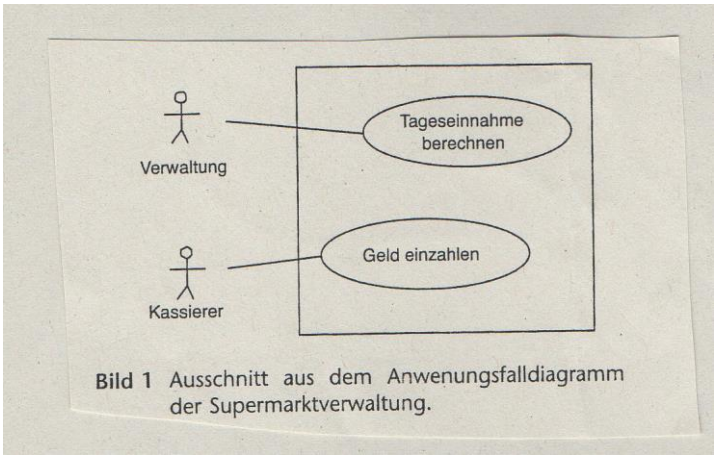
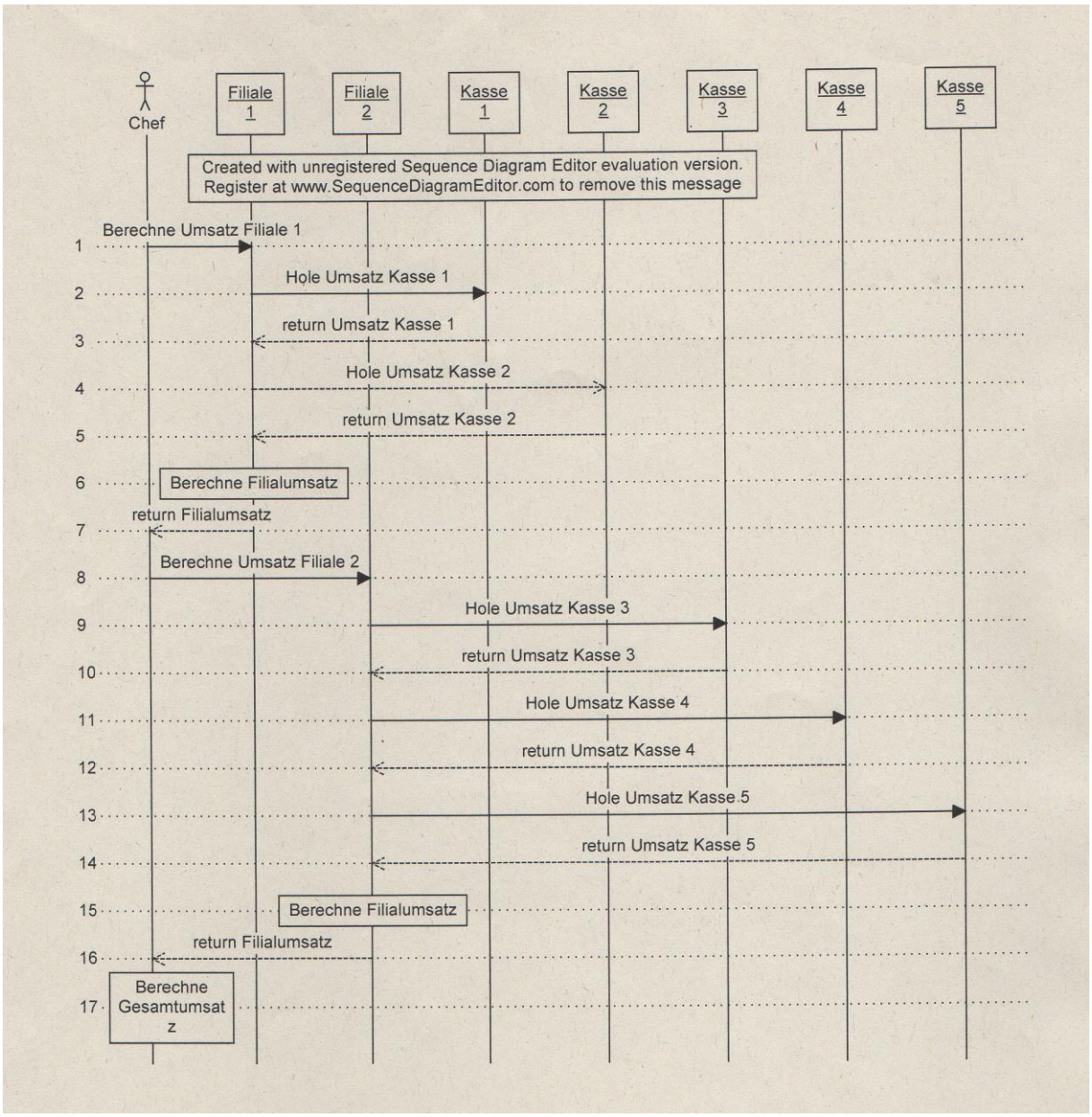


Bild 1 Ausschnitt aus dem Anwenungsfalldiagramm der Supermarktverwaltung.

Sequenzdiagramm für den Anwendungsfall „Gesamtumsatz“ mit 2 Filialen und insgesamt 5 Kassen (Umsetzung des Objektdiagramms)



Fallbeispiel 13 - Autovermietung:

Teil A:

```
public class auto {
  private String Kennzeichen;
  private String Marke;
  private String Farbe;
  private double Verbrauch;
  private double kw;
  private int kmStand;
  private double maxTankinhalt;
  private double aktTankinhalt;

  // Anfang Ereignisprozeduren
  public auto(String k, String m, String f, double v, double kw, int km, double mt, double at)
  {
    this.Kennzeichen=k;
    this.Marke=m;
    this.Farbe=f;
    this.Verbrauch=v;
    this.kw=kw;
    this.kmStand=km;
    this.maxTankinhalt=mt;
    this.aktTankinhalt=at;
  }
  public String getKennzeichen() {
    return Kennzeichen;
  }

  public void setKennzeichen(String Kennzeichen) {
    this.Kennzeichen = Kennzeichen;
  }

  public String getMarke() {
    return Marke;
  }

  public void setMarke(String Marke) {
    this.Marke = Marke;
  }

  public String getFarbe() {
    return Farbe;
  }

  public void setFarbe(String Farbe) {
    this.Farbe = Farbe;
  }

  public double getVerbrauch() {
    return Verbrauch;
  }

  public void setVerbrauch(double Verbrauch) {
    this.Verbrauch = Verbrauch;
  }
```

```
  public double getKw() {
    return kw;
  }

  public void setKw(double kw) {
    this.kw = kw;
  }

  public int getKmStand() {
    return kmStand;
  }

  public void setKmStand(int kmStand) {
    this.kmStand = kmStand;
  }

  public double getMaxTankinhalt() {
    return maxTankinhalt;
  }

  public void setMaxTankinhalt(double maxTankinhalt) {
    this.maxTankinhalt = maxTankinhalt;
  }

  public double getAktTankinhalt() {
    return aktTankinhalt;
  }

  public void setAktTankinhalt(double aktTankinhalt) {
    this.aktTankinhalt = aktTankinhalt;
  }
  public void ausgabe()
  {
    System.out.println("Kennzeichen:    "+this.Kennzeichen);
    System.out.println("Marke:          "+this.Marke);
    System.out.println("Farbe:          "+this.Farbe);
    System.out.println("Verbrauch:      "+this.Verbrauch);
    System.out.println("kw:             "+this.kw);
    System.out.println("Kilometerstand: "+this.kw);
    System.out.println("Tankinhalt max: "+this.maxTankinhalt);
    System.out.println("Tankinhalt akt: "+this.aktTankinhalt);

  }
    // Ende Ereignisprozeduren
}


public class auto1
{
  public static void main(String[] args)
  {
          //2 Instanzen der Klasse auto erzeugen
          auto a1 = new auto("GI-JP 111","Renault","rot",6.2,90,32500,60,32);
          auto a2 = new auto("GI-VH-200","VW","schwarz",7.5,75,65000,50,40);
          //Verändern einiger Attribute
          //Werte anzeigen
```

```
        System.out.println("\nAusgabe a1\n");
        a1.ausgabe();
        System.out.println("\nAusgabe a2\n");
        a2.ausgabe();
  }
}
```

```
public class lkw extends auto {

 // zusätzliche Attribute
 private int maxLadegewicht;      //maximal zulässige Zuladung
 private int aktLadegewicht;      //aktuelles Ladegewicht

 //Methoden
 public lkw(int max,int akt)       //Konstruktor
 {
   this.maxLadegewicht=max;
   this.aktLadegewicht=akt;
 }
 public void setMaxLadegewicht(int kg) {
   this.maxLadegewicht = kg;
 }
 public int getMaxLadegewicht()   {
   return this.maxLadegewicht;
 }

 public void setAktLadegewicht(int kg) {
   this.aktLadegewicht = kg;
 }

 public int getAktLadegewicht() {
   return this.aktLadegewicht;
 }

 public void beladen(int kg) {
   if(this.aktLadegewicht+kg>this.maxLadegewicht)
   {
      this.aktLadegewicht=this.maxLadegewicht;
   }
   else
   {
     this.aktLadegewicht = this.aktLadegewicht+kg;
   }
 }
 public void ausgabe()
 {
       System.out.println("Maximales Gewicht: "+this.maxLadegewicht);
       System.out.println("Aktuelles Gewicht: "+this.aktLadegewicht);
 }
}

public class auto2
{
 public static void main(String[] args)
 {
         //1 Instanzen der Klasse lkw erzeugen
         lkw l1 = new lkw(30000,12000);
         //Werte anzeigen
         System.out.println("\nAusgabe l1\n");
         l1.ausgabe();
 }
}
```

Teil B:

```
public class Kunde {

  // Anfang Variablen
  private int Kdnr;
  private String KName;
  private Buchung meineBuchung;
  // Ende Variablen

  // Anfang Ereignisprozeduren
  public Kunde(int k, String n)
  {
    this.Kdnr=k;
    this.KName=n;
  }

  public int getKdnr() {
    return Kdnr;
  }

  public void setKdnr(int Kdnr) {
    this.Kdnr = Kdnr;
  }

  public String getKName() {
    return KName;
  }

  public void setKName(String KName) {
    this.KName = KName;
  }
  public void buchen(int bnr, String kz,int d)
  {
    Buchung b1=new Buchung(bnr,kz,10);
    this.meineBuchung=b1;
  }
  public Buchung getMeineBuchung() {
    return meineBuchung;
  }

  public void setMeineBuchung(Buchung meineBuchung) {
    this.meineBuchung = meineBuchung;
  }

  // Ende Ereignisprozeduren
}
```

```
public class Buchung {

  // Anfang Variablen
  private int bnr;
  private String kennz;
  private int Dauer;
  // Ende Variablen

  // Anfang Ereignisprozeduren
  public Buchung(int b, String k, int d)
  {
    this.bnr=b;
    this.kennz=k;
    this.Dauer=d;
  }
  public int getBnr() {
    return bnr;
  }

  public void setBnr(int bnr) {
    this.bnr = bnr;
  }

  public String getKennz() {
    return kennz;
  }

  public void setKennz(String kennz) {
    this.kennz = kennz;
  }

  public int getDauer() {
    return Dauer;
  }

  public void setDauer(int Dauer) {
    this.Dauer = Dauer;
  }

  // Ende Ereignisprozeduren
}

import java.io.*;
public class auto3   //Anwendungsfall Kunde bucht Fahrzeug
{
  public static void main(String[] args) throws IOException
  {
     //2 Instanzen der Klasse auto erzeugen
     auto a1 = new auto("GI-JP 111","Renault","rot",6.2,90,32500,60,32,40);
     auto a2 = new auto("GI-VH 200","VW","schwarz",7.5,75,65000,50,40,35);
     //1 Kunden erzeugen
     Kunde k1 = new Kunde(100,"Meier");
```

```
    System.out.println("Guten Tag, Herr "+k1.getKName()+"! Es stehen folgende Fahrzeuge
zur Verfügung:\n ");
    System.out.println(a1.getMarke()+", Kennzeichen: "+a1.getKennzeichen()+"
"+a1.getKw()+" kw");
    System.out.println(a2.getMarke()+", Kennzeichen: "+a2.getKennzeichen()+"
"+a2.getKw()+" kw");
    System.out.print("\nWelches Fahrzeug wollen Sie buchen? (Kennzeichen eingeben): ");
    String kz=Console.s();
    System.out.print("\nWie lange wollen Sie buchen? (in Tagen): ");
    int d=Console.i();
    k1.buchen(1,kz,d);    //Kunde bucht Fahrzeug und erzeugt Buchungsobjekt
    //Werte anzeigen
    System.out.println("\nKundennummer: "+k1.getKdnr()+" Name: "+k1.getKName());
    System.out.println("Hat gebucht: "+k1.getMeineBuchung().getKennz()+" für
"+k1.getMeineBuchung().getDauer()+" Tage");
  }
}
```