

# Softwareprojekt 2014

## Q1

### August-Hermann-Francke – Gymnasium

---



**Gruppe:** \_\_\_\_\_

**Teilnehmer:**

Projektmanager: \_\_\_\_\_

Präsentierer: \_\_\_\_\_

Protokollant: \_\_\_\_\_

Zeitwart: \_\_\_\_\_

Ermutiger: \_\_\_\_\_

Kritikier: \_\_\_\_\_

1	Die Anforderungen – Der Supermarkt .....	3
1.1	Kundenanforderung .....	3
3	Hinweise zur Projektarbeit Supermarkt .....	4
4	Teamarbeit während des Projektes .....	5
5	Phasen des Softwareprojektes .....	6
6	Zusatzaufgaben .....	7
7	Hinweise und Hilfen .....	8
7.1	Hinweis zur Verwendung der Dokumentation .....	8
7.2	Beispiel 1 – Durchlauf einer Liste .....	8
7.3	Hinweis zur Ausgabe einer Queue .....	8
7.4	Hinweise zur verzögerten Ausführung eines Programms .....	9
7.5	Hinweise zur Ausgabe von Informationen in der Konsole .....	9
7.6	Hinweis zu der Erstellung des Entwurfsdiagrammes .....	10
8	Hinweise zur Erstellung der Methoden für die Klasse Supermarkt .....	11
8.1	Beschreibung der Klasse Supermarkt.....	11
8.2	Eine Runde der Simulation .....	12
9	Hinweise zur Erstellung der Klasse Kunde.....	13
9.1	Beschreibung der Klasse Kunde .....	13
9.2	Die Wartezeit des Kunden.....	14
10	Hinweise zur Erstellung der Klasse Kasse .....	15
10.1	Beschreibung der Klasse Kasse.....	15
10.2	Hinweis zu der Generation von Kunden.....	16
11	Anhang.....	17
11.1	Anhang – Objekt- und Klassendiagramme in UML.....	17
11.2	Anhang – Aktivitätsdiagramme .....	18
11.3	Anhang - Entwicklungsdiagramm und Implementationsdiagramm.....	19
11.4	Anhang – Verknüpfung von Klassen und Datensammlungen .....	20
11.5	Anhang – Wiederholung Vererbung.....	21
11.6	Anhang – Java Stylekonventionen für die Programmierung.....	22

Projektdateien und die Gruppeninformation finden sich auf

<http://www.andreas-thiessen.de/ahf/>

IF Q1



## 1 Die Anforderungen – Der Supermarkt

Für eine Supermarktkette soll eine Simulation der Kassenauslastung implementiert werden. Ziel ist es noch vor dem Bau eines neuen Supermarktgebäudes die passende Anzahl von Kassen zu planen um eine Unter- oder Überbelastung zu vermeiden.

Für die Simulation wurden folgende Vorgaben durch den Kunden geäußert (Basisversion):



### 1.1 Kundenanforderung

Es soll ein Augenmerk auf die Kassen und die zugehörigen Warteschlangen vor der Kasse gelegt werden. Für eine Basisversion sollen alle Ausgaben zunächst in der BlueJ-Konsole ausgegeben werden. Es soll möglichst übersichtlich dargestellt werden, wie viele Kassen zur Verfügung stehen und wie sich die Schlangen vor der Kasse entwickeln. Die Simulation soll darüber hinaus in der Basisversion zunächst durch Runden simuliert werden. In einer Runde stellen sich neue Kunden an, und es werden jeweils die vordersten Kunden an den Kassen bearbeitet. Die Schlangen vor den Kassen werden durch die Kunden beschrieben.

#### Kunden

Ein Kunde hat eine Anzahl von Artikeln und kennt seine Wartezeit, die er an der Kasse steht. Diese Wartezeit soll nach jeder Runde angegeben werden. Eine beliebige Anzahl von Kunden soll jede Runde zufällig erzeugt werden. Die Kunden werden von der Simulation an die kürzeste Warteschlange angestellt. Die Kunden sollen automatisch generiert werden.



#### Kassen

Die Kassen können besetzt sein, müssen es aber nicht. An einer Kasse stellen sich beliebig viele Kunden an. Jede Kasse hat einen Namen und weiß wie lang die aktuelle Schlange der Kunden an der Kasse ist. Eine Kasse hat eine gewisse Geschwindigkeit, mit der sie die Artikel eines Kunden abarbeitet (mindestens einen Artikel pro Runde). Die Kasse liefert eine Methode, mit der sie den Kassennamen und alle anstehenden Kunden ausgeben kann.



### 3 Hinweise zur Projektarbeit Supermarkt

Ein Softwareprojekt ist etwas, das nach speziellen Schritten durchgeführt werden sollte. In der Wirtschaft werden diese Schritte verwendet um die Qualität der Software sicherzustellen. Im Unterricht ist es wichtig den Verlauf der Projektarbeit zu notieren, damit sie später transparenter bewertet werden kann. Das Softwareprojekt soll in selbstständiger Arbeit durch die Gruppe durchgeführt werden. Der Lehrer steht nur als letzte Station bei völliger Hilflosigkeit zur Verfügung.

#### **Dauer und Rhythmus**

Das Projekt wird für die Dauer von drei Wochen durchgeführt. Es gibt in dieser Zeit keine Hausaufgabe, es wird jedoch erwartet, dass auch zu Hause an dem Projekt gearbeitet wird. Die gemeinsamen Stunden sollen insbesondere dem Austausch und dem gemeinsamen Arbeiten dienen. In jeder Doppelstunde gibt es eine gemeinsame Besprechung mit dem Kurs, indem Erfahrungen der Gruppen und Fragen an den gesamten Kurs ausgetauscht werden können.

**Wöchentlich**(3x)sollen deshalb folgende Dokumente an den Lehrer geschickt werden:

- Wochenplan (Milestones)
- Aufgabenverteilung + Maßnahmen zur Erfüllung der zugewiesenen Rolle

**Einmalig** soll folgendes Dokument an den Lehrer geschickt werden:

- Problemanalyse + Aktivitätsdiagramme (Es dürfen gerne alle Versionen nummeriert beigelegt werden, damit die Entwicklung noch deutlicher wird)
- Zusammengefasste Milestones und Aufgabenverteilungen
- Eigene Bewertung der Software (mögliche Verbesserungsideen oder Schwächen der Software)
- Dokumentiertes Softwareprojekt als Zip-Datei
- Feedback für den Lehrer (Kritik und Verbesserungsvorschläge für die nächste Durchführung)

#### **Präsentation der Ergebnisse**

Die Projekte der Gruppen werden in einem 7-Minütigen Vortrag dem Kurs vorgestellt. Gehen Sie davon aus, dass die anderen Schüler die Beschreibung der Aufgabenstellung kennen und erläutern Sie Ihre Ausgabe, wichtige Entscheidungen und Besonderheiten Ihres Projektes. Es sollten höchstens zwei Personen an der Vorführung beteiligt sein. Der Kurs erhält nach dem Vortrag noch die Möglichkeit Fragen an die Gruppe zu stellen und sich evtl. Teile des Projektes näher erläutern zu lassen.

#### **Bewertung**

Das *Gruppenprojekt* wird anhand folgender Kriterien bewertet:

- Korrektheit der Software (20%)
- Umfang der Software (20%)
- Dokumentation des Projektes (20%)
- Präsentation (20%)
- Zusammenarbeit im Team (20%)

#### *Einzelleistungen*

Während des Projektes werden fünfminütige mündliche Abfragen zum eigenen Softwareprojekt und dem Thema Abstrakte Datenstrukturen gestellt.



## 4 Teamarbeit während des Projektes

Die Gruppenarbeit zeichnet sich dadurch aus, dass ein Produkt von der Gruppe gemeinsam erarbeitet und kontrolliert wird. Das bedeutet nicht, dass alle Aufgaben gemeinsam erledigt werden müssen. Oft ist es nötig die größere Aufgabenstellung zu zerlegen und in kleinere Aufgaben zu zerlegen, die dann von den einzelnen Gruppenmitgliedern in Einzel- oder Partnerarbeit gelöst werden können. Die einzelnen Teilergebnisse sollten dann wieder von der gesamten Gruppe zusammengefügt und bewertet werden.

In einer Gruppenarbeit ist es sinnvoll Rollen zu verteilen, um den reibungslosen Ablauf der Gruppenarbeit zu garantieren. Je nach Anzahl der Gruppe sollten die Rollen entsprechend der Reihenfolge verteilt werden: Projektmanager, Protokollant, Präsentierer, Zeitwart, Ermutiger, Kritiker.

<p><b>Projektmanager</b></p>  <p>übernimmt die Verantwortung für das Projekt. Verteilt die Aufgaben und Rollen und hilft später die Teilergebnisse zusammenzufügen. Alle Aufgaben werden aber in Absprache mit der Gruppe getroffen. Der Projektmanager ist kein Diktator, sondern eher jemand, der einen Überblick des Geschehens hat.</p>	<p><b>Protokollant</b></p>  <p>zeichnet alle Entscheidungen, die Verantwortlichen, Absprachen und Ergebnisse auf und stellt sie später der gesamten Gruppe zur Verfügung (Email oder Handout).</p>
<p><b>Präsentierer</b></p>  <p>stellt die Ergebnisse der Gruppe vor. Ergebnisse und Erläuterungen zum Vorgehen sollen angemessen wiedergegeben werden. Absprachen mit dem Projektmanager und den anderen Gruppenmitgliedern sind zwingend nötig!</p>	<p><b>Zeitwart</b></p>  <p>hat die zur Verfügung gestellte Zeit im Blick und erinnert die Gruppe insbesondere in der frühen Phase der Gruppenarbeit an ein zügiges Vorgehen. Regelmäßig gibt der Zeitwart Auskunft über die verbleibende Zeit und fragt nach Prognosen der Fertigstellung.</p>
<p><b>Ermutiger</b></p>  <p>hat das Ziel vor Augen und hilft Perfektionisten, etwas gelassener zu arbeiten und ermutigt Faulenzer, an dem Gruppenergebnis teilzuhaben.</p>	<p><b>Kritiker</b></p>  <p>hinterfragt die Vorgehensweise und denkt noch einmal quer. Überlegt Alternativen und versucht die Gruppe durch mögliche Fragen auf die Präsentation vorzubereiten.</p>



## 5 Phasen des Softwareprojektes

### Phase 1 – Gruppenbildung

1. Bilden Sie Gruppen mit höchstens sechs Schülern oder Schülerinnen. Jeder aus der Gruppe übernimmt eine spezielle zusätzliche Aufgabe (siehe Gruppenrollen).
2. Es wird immer in Zweierteams programmiert.
3. Aufgaben sollten immer möglichst gleichmäßig aufgeteilt und Lösungen stets miteinander geteilt werden.

Passen Sie die Zusatzaufgaben an den Charakter der Person an. Jemand der unordentlich ist, sollte nicht Protokollant sein, außer er/sie möchte diese Herausforderung annehmen, um den eigenen Charakter auszubilden.

Für die Projektleitung sollte jemand gewählt werden, der sich gut mit Programmierung auskennt, um den Aufwand einzelner Aufgaben abschätzen zu können. Der Projektleiter / Die Projektleiterin sollte zunächst immer Aufgaben verteilen, und sich als letztes einer Aufgabe zuordnen.

### Phase 2 – Problemanalyse

1. Stellen Sie die obige Beschreibung des Problems mit einem Entwurfsdiagramm (Diagramm mit Klassen ohne Festlegung einer Programmiersprache) dar.
2. Erstellen Sie ein *Aktivitätsdiagramm* für die Ausführung einer Runde.

In der zweiten Phase sollte zunächst die Aufgabenstellung klar und einheitlich in der Gruppe besprochen werden. Ein Klassendiagramm ist ideal um einen Überblick der Problemstellung zu erhalten. Um Klassen extrahieren zu können eignet sich eine Textanalyse. Substantive (Wörter, die für sich selbst stehen können) bilden häufig Klassen. Verben bilden häufig Methoden einer Klasse. Darüber hinaus müssen evtl. noch weitere Hilfsklassen oder Methoden im Klassendiagramm notiert werden. Seien Sie gerade zu Anfang nicht zu perfektionistisch, sondern ergänzen Sie das Klassendiagramm später noch um weitere Methoden und Klassen.

Ein weiterer wichtiger Bestandteil sind die Klassen, die bereits implementiert werden. Zur Problemanalyse gehört deshalb auch die Analyse der vorhandenen Werkzeuge.

Als digitales Visualisierungswerkzeug kann StarUML, AgroUML oder Umbrella auf dem Heimrechner installiert werden. In der Schule können Notizblöcke als Zeichnungs- und Besprechungshilfen dienen.

### Phase3 – Implementierung

1. Besprechen Sie die Schnittstellen (Methoden) der einzelnen Klassen und welche Parameter und Rückgabe-Werte benötigt werden. Notieren Sie *Sequenzdiagramme*, die darstellen welche Methoden aufgerufen werden.
2. Implementieren Sie parallel an den entsprechenden Stellen. Fügen Sie nach Möglichkeit nur getesteten Quelltext zusammen. Verwenden Sie hierfür eine „Hauptversion“, die regelmäßig aktualisiert und an die anderen Gruppenmitglieder verteilt wird.

Die Vorgaben des Klassendiagrammes sind maßgeblich für die Implementierung. Nur so können Aufgaben sinnvoll verteilt werden. Die Aktivitätsdiagramme helfen bei der Entwicklung einzelner Methoden. Durch Sequenzdiagramme werden der Ablauf und die Verknüpfung der verschiedenen Klassen deutlich. Für die parallele Arbeit an Klassen ist es notwendig nicht nur den Aufruf einer Methode, sondern auch die Rückgabewerte festzulegen. Ein Sequenzdiagramm bezieht sich immer auf ein dazugehöriges Aktivitätsdiagramm und sollte deshalb ähnlich benannt werden.



Vergessen Sie in dieser Phase nicht die Klassen-, Aktivitäten- und Sequenzdiagramme zu aktualisieren, falls bei der Implementierung deutlich wird, dass eine andere Vorgehensweise gewählt werden sollte.

## Phase 4 – Evaluation

1. Nach jedem Zusammenfügen (Milestones) sollte die Software zunächst getestet werden, um die nächsten Schritte planen und besprechen können.
2. Wiederholen Sie die Schritte 2-3 bis das Projekt vollständig ist.

Die vierte Phase ist eng verwoben mit der dritten Phase und soll einen Überblick über aktuelle Probleme und Aufgaben liefern. Nehmen Sie sich regelmäßig Zeit um Ihren Fortschritt mit den geplanten Zielen zu vergleichen und ändern Sie gegebenenfalls Ziele oder Arbeitsweisen und Aufgabenverteilungen.

## 6 Zusatzaufgaben

Zusatzaufgaben können selbstständig bearbeitet werden. Die Bearbeitung der Zusatzaufgaben ist insbesondere für eine hervorragende Benotung notwendig. Die Zusatzaufgaben sind optional und es müssen **nicht alle** Zusatzaufgaben bearbeitet werden! Sie sollten vor jeder in Anspruch genommenen Zusatzaufgabe eine lauffähige Zwischenversion sichern und Ergänzungen stets an Kopien durchführen.

### [Z1a]Englisches System:

In der Simulation soll zeitgleich das englische System für Warteschlangen angezeigt werden. Es gibt nur eine Schlange, von der aus die Kunden an die einzelnen Kassen gehen. So steht an einer Kasse immer nur eine Person. Ändern Sie nur die Simulation entsprechend. Fügen Sie eine Warteschlange ein und lassen Sie in jeder Runde alle freien Kassen zunächst mit Kunden aus der Schlange besetzen und direkt bearbeiten.

### [Z1b]Statistiken

Um die Systeme besser vergleichen zu können soll die durchschnittliche Wartezeit der Kunden berechnet werden. Jeder bediente Kunde soll hierfür zusätzlich in einer Liste gespeichert werden, in der die Anzahl der Artikel und die Wartezeit vorhanden sind. Nutzen Sie diese Informationen um durchschnittliche Wartezeiten für die Kunden mit ähnlicher Artikelanzahl zu bestimmen.

### [Z2] WarteZEIT statt WarteRUNDEN

Um die konkrete Wartezeit berechnen zu können sollen die Kunden, sobald sie an einer Kasse „auf die Uhr sehen“ und die Startwartezeit bestimmen. Die Kassen bearbeiten einen Kunden eine gewisse Zeit, der Artikelanzahl entsprechend. Hinweis: Die Kassen sollten als Thread implementiert sein, der von der Simulation gestartet wird. Bei der Bearbeitung kann die Kasse dann für die entsprechende Zeit pausiert werden. Die Zeit kann über die aktuelle Computerzeit abgerufen werden.

## 7 Hinweise und Hilfen

### 7.1 Hinweis zur Verwendung der Dokumentation

Die Klassen List, Queue und Stack sind bereits vorgegeben und sollen nicht mehr verändert werden. Neben dem Quelltext der Klassen steht die Dokumentation der zur Verfügung stehenden Methoden bereit. Die Dokumentation kann über die index.html (doc-Unterverzeichnis) im Browser **oder** durch Doppelklick auf die jeweilige Klasse aufgerufen werden.

Sie sollten als Gruppe für eine erfolgreiche Verwendung der Klassen die entsprechenden Methoden verstanden haben.

Constructor Summary	
<a href="#">List()</a>	Eine leere Liste wird erzeugt.
Method Summary	
void <a href="#">append(Object pObject)</a>	Ein neues Objekt pObject wird am Ende der L
void <a href="#">concat(List pList)</a>	Die Liste pList wird an die Liste angehaengt.
<a href="#">Object</a> <a href="#">getObject()</a>	Falls es ein aktuelles Objekt gibt (hasAccess()

### 7.2 Beispiel 1 – Durchlauf einer Liste

Um eine Liste zu durchlaufen wird eine Wiederholungsstruktur benötigt, die abfragt ob auf das aktuelle Element zugegriffen werden kann (Zeile 2). Doch bevor dies geschehen kann, muss die Liste zunächst auf das erste Element gesetzt werden (Zeile 1). Alle Objekte die in der Liste verwaltet werden. haben den Typ Object. Jedoch hat die Klasse Object (von der alle Klassen erben) nicht die speziellen benötigten Methoden unseres Problemfalles. Deshalb muss ein Typcast durchgeführt werden (Zeile 3). Mit dem erzeugten Schüler können nun alle Methoden die benötigt werden auch angewendet werden (Zeile 4). Um zum nächsten Element zu gelangen, muss nun der aktuelle Zeiger der Liste auf das nächste Element gesetzt werden (Zeile 5). Wird die fünfte Zeile vergessen entsteht eine Endlosschleife, die nur durch Unterbrechung der Java Virtual Machine beendet werden kann (Abbildung 1).

```
1  schülerListe.toFirst();
2  while(schülerListe.hasAccess()){
3      Schüler aktuellerSchüler = ((Schüler)schülerListe.getObject()); //Hier muss ein TypeCast
                                                                    //durchgeführt werden.
4      aktuellerSchüler.ausgeben();
5      schülerListe.next();
6  }
```

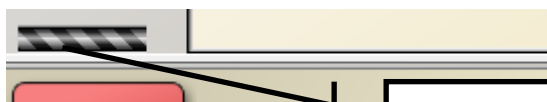


Abbildung 1

Rechtsklick auf den Balken ermöglicht den Reset der Virtual Machine

### 7.3 Hinweis zur Ausgabe einer Queue

Im Gegensatz zu einer Liste kann man eine Queue nicht durchlaufen. Für die Ausgabe muss also beachtet werden, dass die Elemente, die aus der Liste genommen werden, zwischengespeichert werden und später wieder in Form einer Queue zur Verfügung stehen müssen.





## 7.4 Hinweise zur verzögerten Ausführung eines Programms

Häufig möchte man bei Programmen nicht nur das Endergebnis sehen, sondern auch die Entwicklung der Lösung bzw. der Simulation. Die Wartemethode kann für die Ausführung des Programmes für die Anzahl der Millisekunden pausieren.

Da das „Schlafenlegen“ des Programmes einen Fehler erzeugen kann, muss dieser Befehl in einem try-catch-Block aufgefangen werden. Falls ein Fehler entsteht wird der Catch-Block ausgeführt und es wird der Fehler: „Fehler beim Warten“ in der Konsole ausgegeben.

```
private void warte(intanzahlMilliSekunden){
try{
Thread.sleep(anzahlMilliSekunden);
} catch (InterruptedException e){ System.out.println("Fehler beim Warten");
}
}
```

## 7.5 Hinweise zur Ausgabe von Informationen in der Konsole

1. Testen Sie folgende Methode in einem einfachen Testprogramm:

```
public void testAusgabe(){
System.out.println("Ich bin Text");
System.out.print("Ich auch, aber irgendetwas fehlt hier!");
System.out.print(" GENAU, ich wollte eigentlich eine Zeile tiefer stehen!");
System.out.println("Dann probiert es doch mal mit etwas mehr Zeilenumbruch!");
System.out.println();
System.out.println("Aha, es geht also auch ohne Parameterübergabe...interessant!");
}
```

2. Beschreiben Sie die Funktion der folgenden Methoden und grenzen Sie insbesondere print und println voneinander ab.

Methode	Beschreibung
<b>System.out.print(String text)</b>	
<b>System.out.println(String text)</b>	
<b>System.out.print ('\f')</b>	löscht die gesamte Konsole gelöscht werden. So kann z.B. der Supermarkt immer oben in der Konsole dargestellt werden.

3. Testen Sie folgende Methode und beschreiben Sie warum die Ausgabe nicht zu sehen ist.  
Notieren Sie, wie der ausgegebene Text vor dem Löschen wenigstens für einen kurzen Zeitraum sichtbar gemacht werden kann.

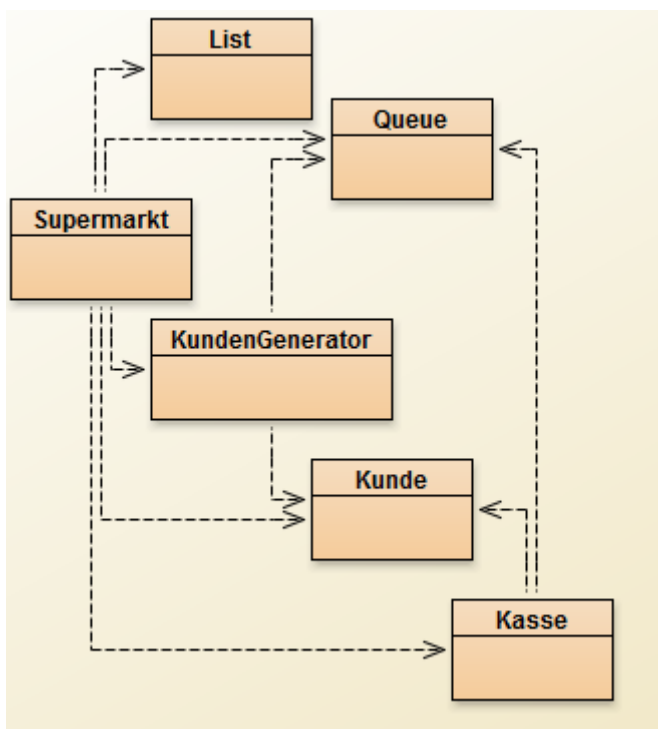
Unsichtbarer Text – Methode	Sichtbarer Text - Methode
<pre>public void schnelleAusgabe(){ System.out.println("Ich bin nur kurz zu sehen"); System.out.print ('\f'); } </pre>	

## 7.6 Hinweis zu der Erstellung des Entwurfsdiagrammes

Ein einfaches Entwurfsdiagramm kann erst einmal nur die Klassen und ihre Assoziationen angeben. In dem folgenden Diagramm sieht man eine Problembeschreibung. Es wird deutlich welche Klassen auf die Liste bzw. die Queue zugreifen.

Die Supermarkt-Klasse steuert den gesamten Ablauf und besitzt deshalb die meisten Assoziationen. Ein Supermarkt hat mindestens eine Kasse und verwaltet die Erzeugung von Kunden und die Kunden selber. Der KundenGenerator hat eine Verknüpfung mit der Kundenklasse und kennt auch die Queue, weil die erzeugten Kunden (sobald es mehrere sind) in einer Queue übergeben werden.

Eine erste Version könnte also wie folgt aussehen. Für ein hilfreicherer Entwurfsdiagramm sollten allerdings noch die Methoden und Attribute hinzugefügt werden.





## 8 Hinweise zur Erstellung der Methoden für die Klasse Supermarkt

Die Klasse Supermarkt soll später für die Simulation aufgerufen werden. Im Idealen Fall kann die gesamte Simulation über diese Klasse verwaltet werden, d.h. die Benutzer sehen die Ausgabe des Supermarktes und können diese Verändern, indem Sie z.B. weitere Kassen hinzufügen oder steuern ob die nächste Runde beginnen soll.

Notieren Sie im nebenstehenden Klassendiagramm die benötigten Attribute und Methoden. Aktualisieren Sie diese Klasse im Laufe des Projektes. Die Methode **naechsteRunde()** und das Attribut **runde** wurden bereits eingefügt, damit der Benutzer über die Methode die nächste Runde ausführen kann. Die Supermarkt-Klasse verwaltet die aktuelle Rundenzahl im Attribut **runde**. Die Klasse Supermarkt sollte darüber hinaus unbedingt eine Ausgabemethode besitzen, die die einzelnen Ausgabemethoden der Kassen aufruft.

Supermarkt
Attribute:  -runde: Zahl
Methoden:  +naechsteRunde()

### 8.1 Beschreibung der Klasse Supermarkt

Erstellen Sie eine Tabelle, die die Funktionalität der Klasse zu den einzelnen Milestones angibt.

Mile-stone	Funktionalität und Methoden	Benötigte Attribute
1		
2		
3		

## 8.2 Eine Runde der Simulation

Die Klasse Supermarkt verwaltet die einzelnen Runden. Die folgenden Schritte sollen Ihnen helfen eine Steuermethode zu schreiben, die wiederum weitere Methoden aufruft. Schreiben Sie deshalb zunächst eine grobe Anordnung und werden Sie dann immer detaillierter.

1. Entwickeln Sie zunächst ein Aktivitätsdiagramm für eine Runde
  - a. Was soll zu Anfang jeder Runde passieren?
  - b. Welche Dinge passieren während der Runde?
  - c. Welche Dinge sollen am Ende der Runde durchgeführt werden?
2. Überlegen Sie, wie eine einzelne Runde mehrmals wiederholt werden kann.
  - a. Die nächste Runde sollte durch den Benutzer aufrufbar sein.
  - b. Es sollte auch eine Methode geben, die eine bestimmte Anzahl von Durchgängen im Parameter erhält und diese verwendet.

Notieren Sie hier das Aktivitätsdiagramm der Runde:



## 9 Hinweise zur Erstellung der Klasse Kunde

Ein Kunde sollte zunächst einmal einige Eigenschaften besitzen, z.B. sollte ein Kunde die Anzahl seiner Waren kennen und auch die Runde in der seine Wartezeit beginnt. Durch Methoden sollten diese Eigenschaften verändert werden können. Darüber hinaus sollte die Kundenklasse eine Methode besitzen, mit der die Anzahl der Waren und evtl. die Startrunde ausgegeben werden können. Es sollte unbedingt eine ausgabe-Methode geben, mit der ein Kunde die eigenen Informationen in der Konsole ausgeben lassen kann.

Ein Kunde sollte keine weiteren Kassen kennen. In einer Simulation ist es zwar wichtig, den Kunden zu modellieren, jedoch muss nicht die reale Wirklichkeit nachgebildet werden. Der Kunde wird an eine Kasse angestellt.

Kunde
<b>Attribute:</b>  -artikel: Zahl
<b>Methoden:</b>

### 9.1 Beschreibung der Klasse Kunde

Erstellen Sie eine Tabelle, die die Funktionalität der Klasse zu den einzelnen Milestones angibt.

Mile-stone	Funktionalität und Methoden	Benötigte Attribute
1		
2		
3		

## 9.2 Die Wartezeit des Kunden.

Der oder die Kundin sollte seine Wartezeit bestimmen können, hierfür benötigt er das Wissen über die aktuelle Runde.

Diese Information kann von einer Klasse, die die Runde verwaltet erhalten werden.

	Für die Methoden, die die Wartezeit berechnen muss
Prinzip	die aktuelle Runde als Parameter übergeben werden.
Deklaration der Klasse mit der Variable	<pre>public class KlasseMitRunde{     int runde;     public KlasseMitRunde(){         runde = 0;     }      //Gibt die aktuelle Runde zurück     public int getRunde(){         return runde;     } }</pre>
Deklaration: Klasse, die die Variable	<pre>public class KlasseDieRundeBenötigt{     public KlasseDieRundeBenötigt(){     }      public void gibDieAktuelleRundeAus( int aktuelleRunde){         System.out.println(aktuelleRunde);     } }</pre>
Verwendung	<pre>public class TestKlasse{     KlasseMitRunde kMR;     KlasseDieRundeBenötigt kDRB;      public TestKlasse(){         kMR = new KlasseMitRunde();         kDRB = new KlasseDieRundeBenötigt();         int aktuelleRunde = kMR.getRunde();         kDRB.gibDieAktuelleRundeAus( aktuelleRunde );     } }</pre>

### 9.2.1 Zusatz:

Eine Alternative zur Parameterübergabe oder einem Attribut Supermarkt in der bildet das Singleton-Pattern. Dabei kann auf die Supermarkt-Klasse zugegriffen werden. Recherchieren Sie „Singleton-Pattern im Zusammenhang von Java“ und bauen Sie dieses Pattern evtl. in Ihr Projekt ein.





## 10 Hinweise zur Erstellung der Klasse Kasse

Die Kasse besitzt neben einer Queue, die die Kunden verwaltet noch einige zusätzliche Informationen. So kann die Kasse z.B. auch die Länge der Schlange verwalten. Die Queue besitzt keine Ausgabe-Methode, diese muss in der Kassens-Klasse implementiert werden. Die Kasse sollte darüber hinaus noch das Anstellen, Bedienen und Entfernen der Kunden berücksichtigen.

Für die Implementierung der Methoden benötigen Sie Wissen zur Verwendung der Queue und der Zugriffsoperationen.

Kasse
Attribute:  -
Methoden:

### 10.1 Beschreibung der Klasse Kasse

Erstellen Sie eine Tabelle, die die Funktionalität der Klasse zu den einzelnen Milestones angibt.

Milestone	Funktionalität und Methoden	Benötigte Attribute
1		
2		
3		

## 10.2 Hinweis zu der Generierung von Kunden

Eine Klasse, die Kunden automatisch generiert, sollte neue Kunden erzeugen und ihnen eine zufällige Anzahl von Artikeln zuweisen. Java besitzt in der Standardbibliothek bereits eine Zufallsfunktion. Die Methode kann über `Math.random()` aufgerufen werden. Die Methode erzeugt jedoch Werte im Zahlenbereich zwischen 0 und 1, wobei die 1 nie zurückgegeben wird.

Die static Methode *random* der Klasse *Math* in Java ist gefolgt definiert:

```
public static double random
```

Returns: a pseudorandom double greater than or equal to 0.0 and less than 1.0

### 10.2.1 Verwendung des Zufallsgenerators:

<b>10.2.2 Zufallszahlen zwischen <math>\geq 0</math> und <math>&lt; 1</math> also <math>[0,1[</math></b> <pre>public static double myRandom() {     return Math.random(); }</pre>	Ausgabe von <i>myRandom()</i> ist z.B.: 0.2558734951799192 0.03857502479001995 0.08562741500057713 0.2329257841571789 0.7471882261881438
<b>10.2.3 Zufallszahlen zwischen <math>\geq 0</math> und <math>&lt; n</math> also <math>[0,n[</math></b> <pre>public static double myRandom(double high) {     return Math.random() * high; }</pre>	Ausgabe von <i>myRandom(3)</i> z.B.: 1.6597912671200072 0.741373630031203 2.657789140634944 1.4622900147468707 2.4057227757910007
<b>10.2.4 Zufallszahlen zwischen <math>\geq x</math> und <math>&lt; y</math>, also <math>[x,y[</math></b> <pre>public static double myRandom(double low, double high) {     return Math.random() * (high - low) + low; }</pre>	Ausgabe von <i>myRandom(2,5)</i> z.B.: 3.94313683944274 4.4750980234929 2.0714770630354 4.8240695985982 3.956272776986
Um ganzzahlige Zufallszahlen zu erstellen ist ein Cast mit Hilfe von ( <i>int</i> ) notwendig.  <b>10.2.5 Ganzzahlige Zufallszahlen zwischen <math>\geq x</math> und <math>&lt; y</math>, also <math>[x,y[</math></b> <pre>public static int myRandom(int low, int high) {     return (int) (Math.random() * (high - low) + low); }</pre>	Ausgabe von <i>myRandom(2,5)</i> z.B.: 3 4 2 2 3
<b>10.2.6 Zufallszahlen inklusive Obergrenze</b> Soll der high-value auch noch mitaufgenommen werden, hilft dieser kleine Trick: <pre>public static int myRandomWithHigh(int low, int high) {     high++;     return (int) (Math.random() * (high - low) + low); }</pre>	Nun ist auch die 5 erhalten; Ausgabe von <i>myRandomWithHigh(2,5)</i> z.B.: 4 3 5 2 2

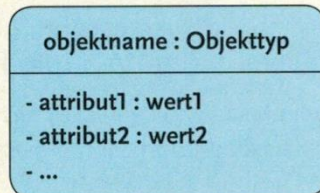


## 11 Anhang

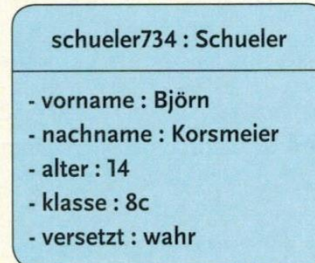
### 11.1 Anhang – Objekt- und Klassendiagramme in UML

#### Das Objektdiagramm

##### Allgemein



##### Beispiel

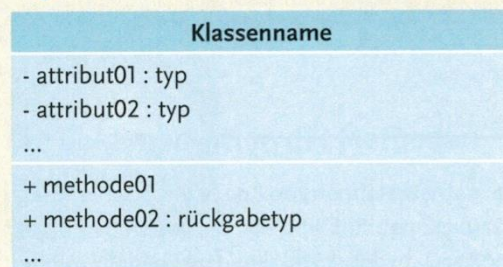


Ein Objektdiagramm benennt in der Kopfzeile das Objekt und gibt seinen Typ („seine Klasse“) an. Die Attribute sind mit konkreten Werten belegt, die in ihrer Gesamtheit den Zustand des Objekts definieren. Dieser Zustand kann sich durch eine Änderung einzelner oder mehrerer Attribute verändern.

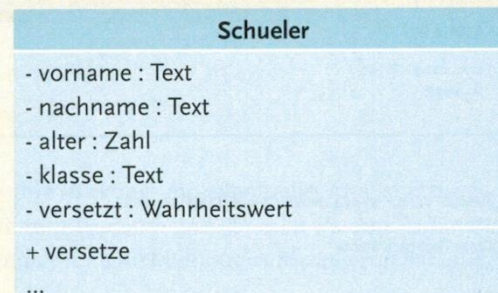
Die Objekte können die Methoden der Klasse anwenden, von der sie instanziiert wurden. Diese werden aber in den Objektdiagrammen nicht angegeben.

#### Das Entwurfsdiagramm

##### Allgemein



##### Beispiel



Ein Entwurfsdiagramm ist noch programmiersprachenunabhängig. Es besitzt schon alle wesentlichen Attribute, deren Typangabe beschränkt sich jedoch auf die Datentypen Text, Zahl und Wahrheitswert.

Gegebenenfalls werden wesentliche Methoden angegeben. Bei Rückgabewerten von Methoden werden auch die Typen Text, Zahl und Wahrheitswert unterschieden. Da Konstruktoren sowie get- und set-Methoden programmiersprachenabhängig sind, werden diese im Entwurfsdiagramm nicht angegeben.



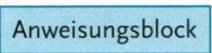

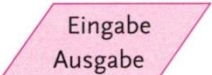
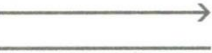
Ein „-“ vor einem Attribut bzw. einer Methode deklariert diese als private, sodass ein Zugriff von Objekten anderer Klassen nicht möglich ist. Ein „+“ deklariert sie als public: der Zugriff von außen ist möglich.



## 11.2 Anhang – Aktivitätsdiagramme

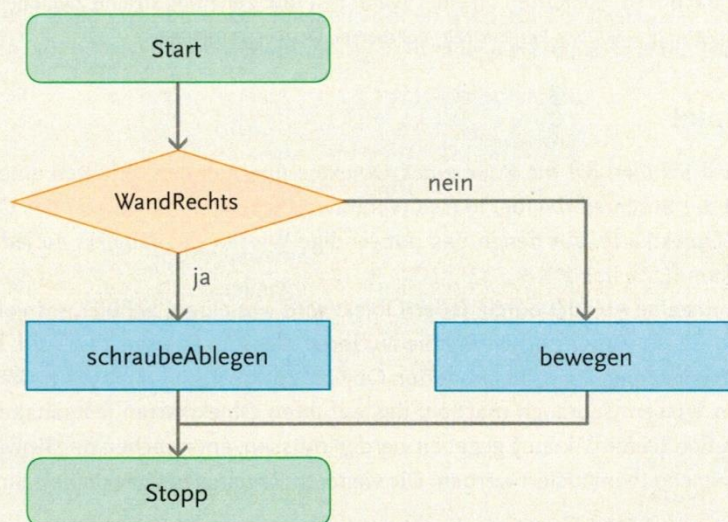
Eine gute Möglichkeit, komplexe Probleme bzw. deren Lösung darzustellen, ist die Benutzung grafischer Unterstützung. Auch in der Informatik bedient man sich dieses Hilfsmittels und erstellt ein klassisches Flussdiagramm in Form eines sogenannten Programmablaufplans (PAP). Dieser kann als Schnittstelle zwischen umgangssprachlicher Formulierung der Lösung und der Programmierung dienen.

Folgende sechs grafische Bestandteile benötigen wir, um einen PAP erstellen zu können:

	Start und Ende des Programms Kann jeweils nur einmal auftauchen
	
	Anweisung oder Anweisungsblock
	Verzweigung mit zwei Ausgängen (ja & nein)
	Eingabe in den Prozess oder Ausgabe
	Verbindungen zum nächstfolgenden Element

Ein Algorithmus beginnt stets mit *Start* und endet mit *Stopp*. Dazwischen werden Anweisungsblöcke, Verzweigungen und Ein- und Ausgaben angeordnet und miteinander verbunden.

Im Beispiel wird je nach Wahrheitsgehalt der Bedingung *wandRechts*, die Methode *schraubeAblegen* oder *bewegen* ausgeführt. Danach ist das Programm mit *Stopp* beendet.

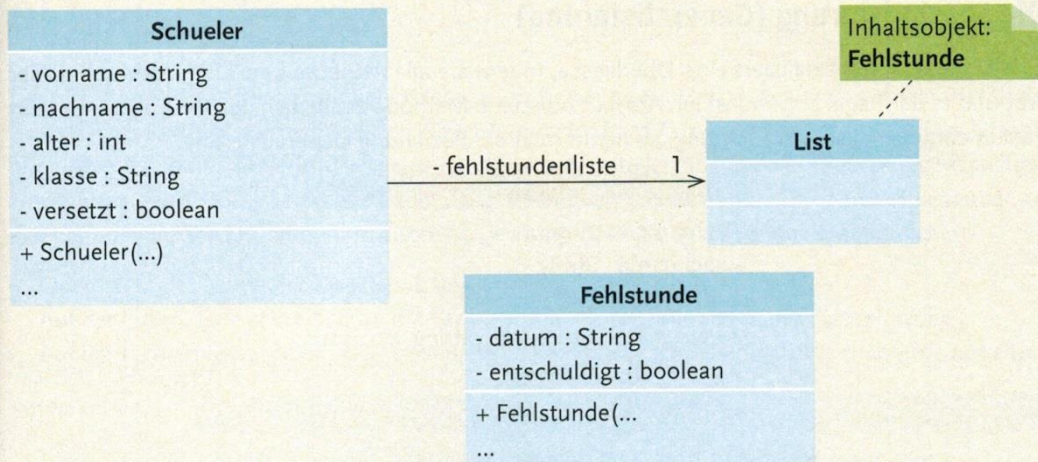


Eine weitere Möglichkeit der grafischen Darstellung von Software ist das Struktogramm, das allerdings für Neueinsteiger komplizierter zu erstellen und zu analysieren ist als ein PAP.

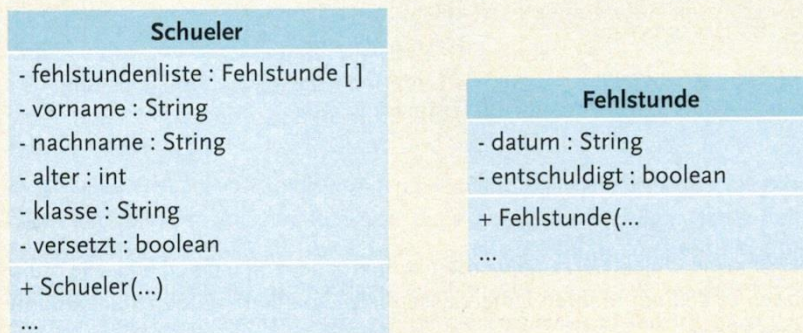




## 11.3 Anhang - Entwicklungsdiagramm und Implementationsdiagramm

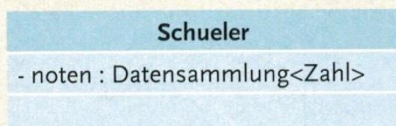


Soll die Datenansammlung mithilfe eines Arrays umgesetzt werden, so kann dies direkt in die Liste der Attribute aufgenommen werden, ohne eine Assoziation zu verwenden:

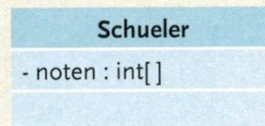


Das gilt insbesondere auch für primitive Datentypen, hier am Beispiel einer Notenliste:

### Entwurfsdiagramm



### Implementationsdiagramm

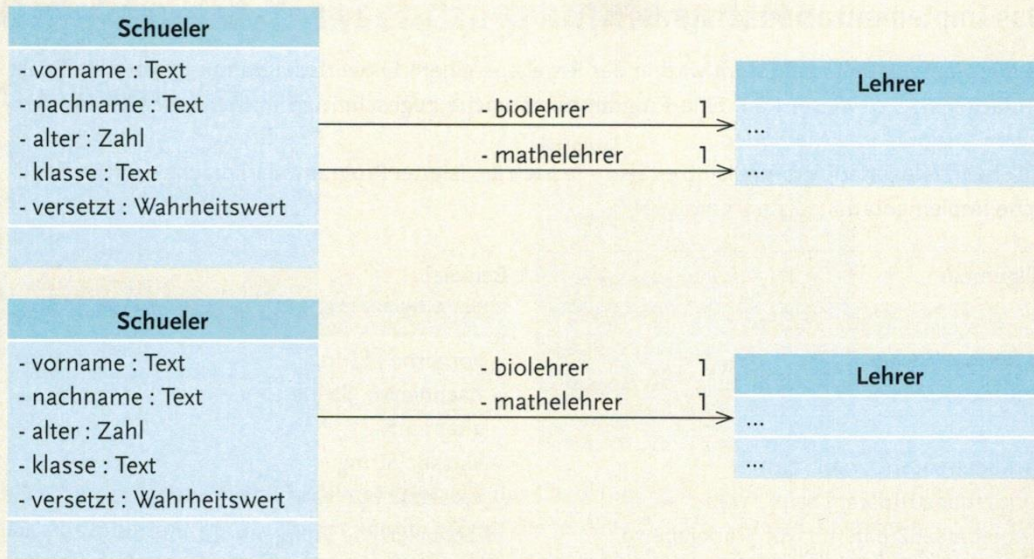


### Multiplizitäten

Multiplizität	
1	Ein Objekt der Klasse assoziiert genau ein Objekt der anderen Klasse.
0..1	Ein Objekt der Klasse assoziiert kein oder ein Objekt der anderen Klasse.
0..*	Ein Objekt der Klasse assoziiert beliebig viele Objekte der anderen Klasse.
m..*	Ein Objekt der Klasse assoziiert mindestens m bis beliebig viele Objekte der anderen Klasse.

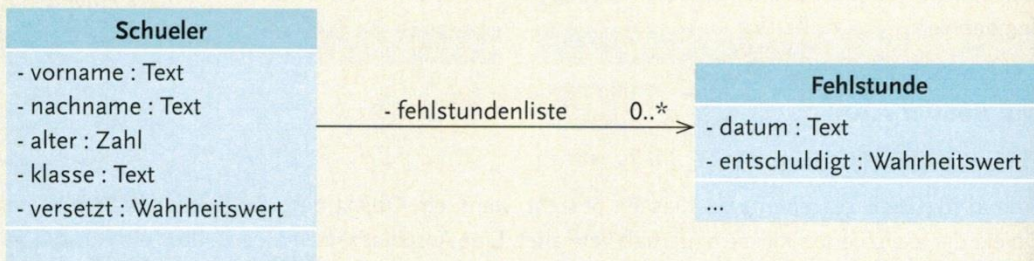


## 11.4 Anhang – Verknüpfung von Klassen und Datensammlungen

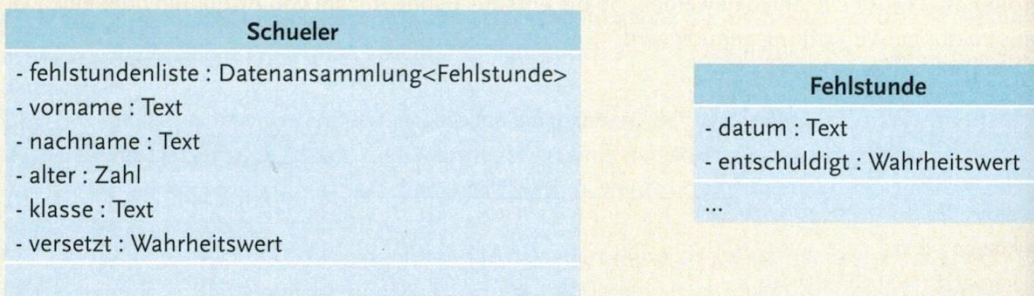


### Assoziationen und Datenansammlungen

Häufig besteht eine Beziehung zu mehreren Objekten einer Klasse. Diese Assoziationen können dann über Datenansammlungen (Datenstrukturen) realisiert werden. Durch das Entwurfsdiagramm wird ersichtlich, dass ein Schüler beliebig viele Fehlstunden ansammeln kann und dass diese Fehlstunden in einer Datenansammlung namens *fehlstundenliste* verwaltet werden sollen.



Anstatt eines Assoziationspfeils kann die Datenansammlung auch direkt in die Klasse Schueler übernommen werden:



Bei der Überführung in ein Implementationsdiagramm muss entschieden werden, welche Datenstruktur die Fehlstunden verwalten soll. Die Assoziation besteht nun zwischen der assoziierenden Klasse und der gewählten Datenstruktur. Die gewünschten Inhaltsobjekte werden über eine Notiz (engl.: note) mit dieser verbunden.

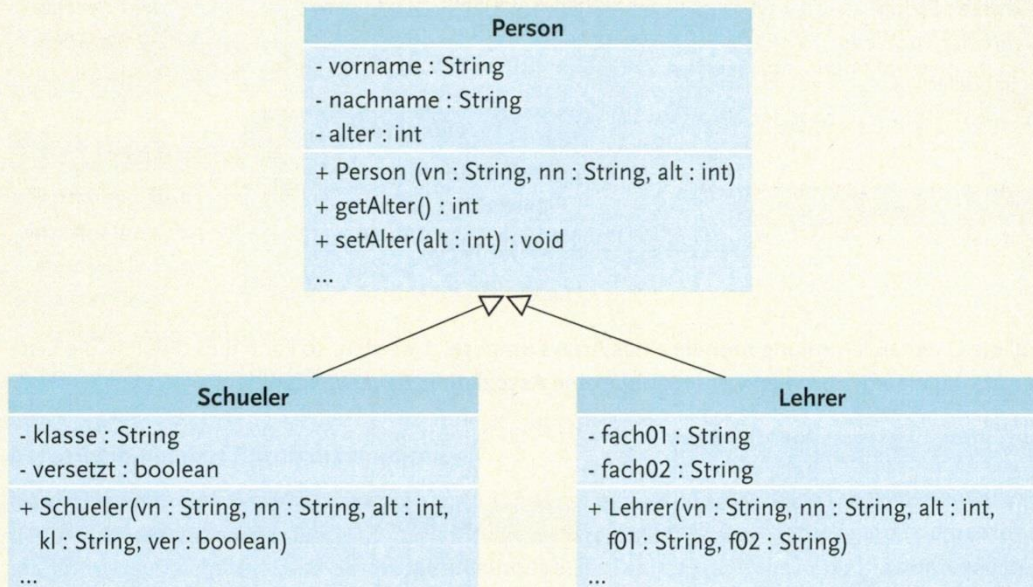




## 11.5 Anhang – Wiederholung Vererbung

### Die Spezialisierung (Generalisierung)

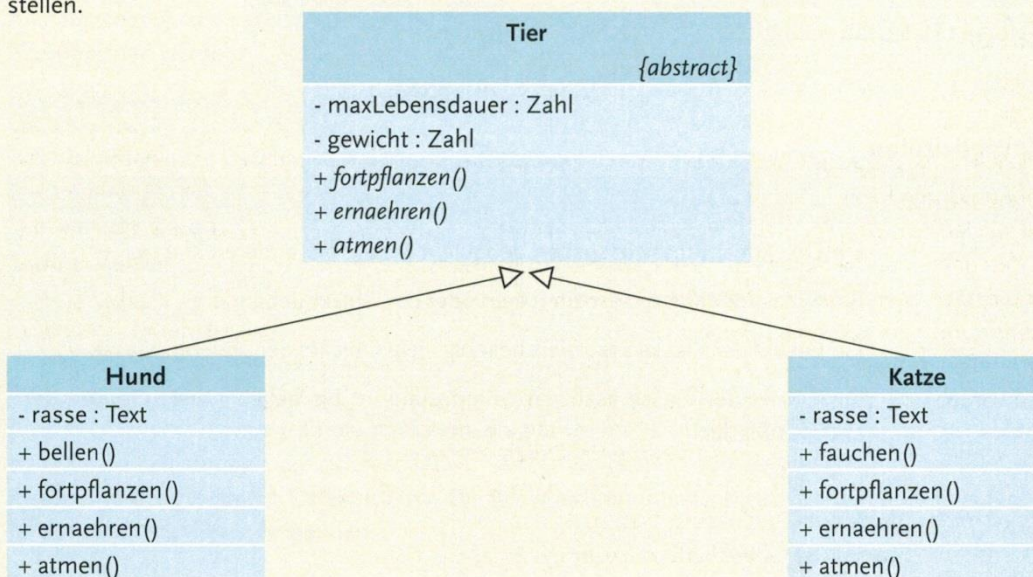
Eine Unterklasse spezialisiert eine Oberklasse, indem sie alle Attribute und Methoden von dieser erbt und in der Regel zumindest ein Attribut oder eine Methode zusätzlich besitzt. Folgt man dem Pfeil in entgegengesetzter Richtung, so nennt man die Beziehung Generalisierung.



### Abstrakte Klassen

Sollen von einer Oberklasse keine Objekte erzeugt werden können oder soll eine Oberklasse exakt vorgeben, welche Methoden unbedingt in ihren Unterklassen implementiert werden sollen, dann benutzt man eine abstrakte Klasse.

Eine Klasse ist abstrakt, wenn eine ihrer Methoden abstrakt ist. Im UML-Diagramm wird dazu das Signalwort *abstract* im Klassennamen zugefügt und die abstrakten Methoden werden kursiv geschrieben. Unterklassen von abstrakten Klassen können wiederum abstrakt sein. Sollen sie hingegen konkret sein, so müssen sie Implementierungen für alle geerbten abstrakten Methoden bereitstellen.





## 11.6 Anhang – Java Stylekonventionen für die Programmierung

Damit der Programmcode immer übersichtlich und damit auch effizient – im Sinne der Wiederverwertung – ist, sollte sich jede Programmiererin/jeder Programmierer an gewisse Stylekonventionen halten. Dadurch erhält man in der Programmiergemeinschaft einen einheitlichen Stil und eine bessere Kompatibilität der einzelnen Programmsequenzen. Andere Programmiererinnen/Programmierer können sich besser in den Code einlesen und ihn besser verstehen.

1. **Namen von Klassen, Methoden, Attributen** etc. werden mit eindeutigen, verständlichen Namen versehen.
2. **Klassen** beginnen immer mit einem Großbuchstaben.

```
public class Example() { ... }
```

3. **Methoden** beginnen mit einem kleinen Buchstaben. Kommen mehrere Wörter in einem Methodennamen vor, so wird jeder Anfangsbuchstabe groß geschrieben.

```
public void methodeWithManyNames() { ... }
```

4. **Attribute** verwenden die gleiche Konvention wie Methoden.

```
int integerValue = 0;
```

5. Bei der **Klammerung** wird – für eine bestmögliche Übersicht – auf das Öffnen und Schließen auf der gleichen Ebene geachtet. Die Einrücktiefe in einer Ebene sollte ebenfalls einheitlich sein.

```
public class FirstFrame extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("FirstFrame");
        g.drawLine(70, 50);
    }
}
```

6. Jedes Programm wird sauber und ausführlich **dokumentiert**: jedes (wichtige) Attribut, jede Methode und auch jede Klasse. (Was bezwecken sie?, Warum brauche ich sie?, etc.) Jede neue Datei wird mit einer „Titelleiste“ versehen:

```
/**
 * @author D.Tepaße
 *
 * Ein Applet mit Variablen und einer for-Schleife.
 * Es wird eine einfache Tabelle Fahrenheit/Celsius im
 * Appletbereich angezeigt.
 */
```