# Eigene Klassen und Objekte in Java

**Einstiegsaufgabe:** Für die Implementierung eines Vokabeltrainers sollen mehrere Vokabeln mit ihrer deutschen und ihrer englischen Bezeichnung in einem Programm gespeichert werden. Zu jeder Vokabel soll außerdem gespeichert werden, ob die Vokabel bereits beherrscht wird und wie viele Versuche zum Lernen dieser Vokabel bereits benötigt wurden.

Diskutieren Sie, wie die Daten im Programm gespeichert werden könnten. Welche Vor- und Nachteile oder Schwierigkeiten gibt es dabei?

## **Das objektorientierte Modell**

Unsere Welt besteht aus Objekten, die miteinander in Interaktion treten. Das können physische Objekte sein, die man anfassen kann, wie z. B. Bücher, Tische oder Personen, oder gedankliche Objekte, wie z. B. Vokabeln oder Nachrichten in einem sozialen Netzwerk. Programme sollen häufig einen Ausschnitt der realen Welt mit ihren Objekten und bestimmten Prozessen abbilden. Objektorientierte Sprachen bieten daher Strukturen an, mit denen sich ein Modell der realen Welt erstellen lässt. Für ein Objekt der realen Welt wird eine spezielle Datenstruktur erstellt, die die relevanten Eigenschaften des Objektes abbildet. Ergänzt werden Operationen, die das Objekt ausführen kann.

In der Regel müssen mehrere gleichartige Objekte der realen Welt abgebildet werden. So werden für den Vokabeltrainer aus der Einstiegsaufgabe ganz viele Vokabeln benötigt. Zu jeder Vokabel müssen die gleichen Eigenschaften (z. B. deutsche Bezeichnung, englische Bezeichnung, Anzahl der Versuche und ob sie bekannt ist) gespeichert werden, auch wenn sich die Vokabeln in den konkreten Werten dieser Eigenschaften unterscheiden. Für eine Menge von gleichartigen Objekten wird daher in Java zunächst eine **Klasse** angelegt, die später als eine Art **Bauplan** für die konkreten Objekte dient. In der Klasse wird festgelegt, welche Variablen die Objekte besitzen, um die Eigenschaften (**Attribute**) abzubilden, und über welche **Operationen** (in Java Methoden genannt) sie verfügen. Die Klasse Vokabel kann z. B. eine Operation pruefen zur Verfügung stellen, die als Parameter zwei Zeichenketten erhält. Die Operation soll überprüfen, ob die übergebenen Werte zu der deutschen bzw. englischen Bezeichnung dieser Vokabel passen. Abbildung 1 stellt den Zusammenhang zwischen der Klasse Vokabel und einzelnen Objekten vom Typ Vokabel anhand einer **Klassenkarte** und mehreren **Objektkarten** dar. Die drei verschiedenen Objekte des Typs Vokabel unterscheiden sich in den Werten der Attribute. Sie besitzen jedoch alle die gleichen Operationen. Deshalb werden in einer Objektkarte zur Vereinfachung häufig nur die Attribute mit ihren konkreten Werten dargestellt. Abbildung 2 zeigt die vereinfachte Darstellung des Objekts v1 vom Typ Vokabel. Ein konkretes Objekt einer Klasse bezeichnet man in Java auch als **Instanz** einer Klasse.



Abbildung : Klassenkarte der Klasse Vokabel mit drei Objektkarten für Objekte des Typs Vokabel



Abbildung : Beispiel für eine vereinfachte Objektkarte ohne Operationen

## Implementierung einer Klasse am Beispiel der Klasse Vokabel

Beispiel 1 auf S. 4 zeigt die Implementierung der Klasse Vokabel[[1]](#footnote-1). Jedes Objekt der Klasse Vokabel soll über die vier **Attribute**, die im Klassendiagramm abgebildet sind, verfügen. Dementsprechend wer­den in der Klasse vier Variablen mit geeigneten Datentypen für die Attribute festgelegt (Zeile 3 bis 6):

* wortD enthält die deutsche Bezeichnung
* wortE enthält die englische Bezeichnung
* versuche enthält die Anzahl der Versuche, wie oft die Variable schon geprüft wurde
* gewusst speichert, ob die Vokabel schon einmal richtig eingegeben wurde

Nach den Attributen folgt in den Zeilen 10 bis 15 die Definition einer speziellen Methode, die genau­so heißt, wie die Klasse, und keinen Rückgabewert hat. Das ist der **Konstruktor** der Klasse. Er wird benötigt, um ein Objekt der Klasse zu erzeugen. In der Konstruktor-Methode wird z. B. definiert, mit welchen Anfangswerten die Variablen für die Attribute belegt werden. Der Konstruktor legt somit einen Start­zustand des Objektes fest. Es kann verschiedene Konstruktoren geben, die alle gleich heißen, sich aber in den Parametern unterscheiden. Im Beispiel werden dem Konstruktor die deut­sche und die englische Bezeichnung übergeben. Diese Werte werden in der Definition des Konstruk­tors den ent­sprechenden Attributen (Variablen) zugewiesen. Da beim Erzeugen eines neuen Objektes vom Typ Vokabel diese Vokabel noch nicht geprüft wurde, erhalten die Attribute versuche und gewusst fest­gelegte Startwerte. Es könnte noch ein zweiter Konstruktor hinzugefügt werden, der neben den Parametern für die deutsche und die englische Bezeichnung einen weiteren Parameter vom Typ Wahrheitswert erhält, der der Variablen gewusst zugewiesen wird. Wird hier der Wert true übergeben, könnte eine neue Vokabel angelegt werden, die von Beginn an als „gewusst“ gekennzeichnet wird.

Abschließend folgen in den Zeilen 18 bis 51 die **Operationen**, die jedes Objekt der Klasse zur Verfügung stellen soll. Allen Variablen für die Attribute wurde das Schlüsselwort **private** voran­gestellt. Das bedeutet, dass von außen der Wert eines Attributs weder direkt verändert noch aus­gelesen werden kann. In den Fällen, in denen ein Auslesen oder Verändern dennoch möglich sein soll, steht deshalb eine entsprechende get- bzw. set-Methode zur Verfügung. Die **get-Methode** gibt den entsprechenden Wert der Variablen zurück. Die **set-Methode** setzt den Wert der Variablen auf den übergebenen Parameter. Dieses Vorgehen gibt dem Programmierer bzw. der Programmie­rerin der Klasse die Kontrolle, ob und wie die Werte der Attribute verändert werden dürfen. So kön­nen in der Klasse Vokabel beispielsweise die Anzahl der versuche und die Variable gewusst nicht beliebig verändert werden. Das Hochzählen der Versuche und das Setzen der Variablen ge­wusst geschehen in der Methode pruefen, abhängig davon, ob die als Parameter übergebenen Werte zu den Werten der Attribute wortD bzw. wortE passen oder nicht. Mithilfe der Methode reset können die Attribute versuche und gewusst auf den Ausgangswert 0 bzw. false zurückgesetzt werden.

Dieses Prinzip bezeichnet man als **Kapselung**: Der Zustand des Objektes kann nur über die Opera­tio­nen verändert werden. Die Operationen bilden die **Schnittstelle** des Objektes nach außen. Verände­rungen am Objekt sind nur über diese Schnittstelle möglich. Dadurch können nachträglich Änderung­en an der internen Struktur einer Klasse vorgenommen werden, z. B. an der Definition einer Opera­tion, ohne dass dies Änderungen in dem Programm nach sich zieht, das die Klasse verwendet. In der Klassenkarte wird dieses Prinzip durch ein Minuszeichen für private vor den Attributen und ein Pluszeichen vor den Methoden für public (öffentlich) dargestellt. Nähere Erläuterungen dazu finden sich im Abschnitt „Sichtbarkeit von Attributen und Operationen“.

1. public class Vokabel{
2. // Anfang Attribute
3. private String wortD;
4. private String wortE;
5. private int versuche;
6. private boolean gewusst;
7. // Ende Attribute
8. //Konstruktor
9. public Vokabel(String d, String e){
10. wortD = d;
11. wortE = e;
12. versuche = 0;
13. gewusst = false;
14. }
16. // Anfang Methoden
17. public void reset(){
18. versuche = 0;
19. gewusst = false;
20. }
21. public boolean pruefen(String d, String e){
22. versuche = versuche + 1;
23. if(wortD.equals(d) && wortE.equals(e)){
24. gewusst = true;
25. return true;
26. }**else**{
27. gewusst = **false**;
28. **return** **false**;
29. }
30. }
32. public String getWortD(){
33. return wortD;}
34. public void setWortD(String wortDNeu){
35. wortD = wortDNeu;}
36. public String getWortE(){
37. return wortE;}
38. public void setWortE(String wortENeu){
39. wortE = wortENeu;}
40. public int getVersuche(){
41. return versuche;}
42. public boolean getGewusst() {
43. return gewusst;}
44. // Ende Methoden
45. }

Beispiel 1: Implementierung der Klasse Vokabel

## Verwenden einer Klasse am Beispiel der Klasse Vokabel

Die Klasse Vokabel kann nun verwendet werden, um in einer Klasse Vokabeltrainer meh­rere Objekte des Typs Vokabel zu erzeugen und beispielsweise in einer Reihung vom Typ Voka­bel zu speichern. Beispiel 2 zeigt einen Ausschnitt des beiliegenden Vokabeltrainer-Pro­gramms (s. Abb. 3). Insbesondere der Quellcode zum Erzeugen der Benutzeroberfläche wird nicht mit aufgeführt.

1. public class Vokabeltrainer extends JFrame {
2. // Anfang Attribute
3. […]
4. private Vokabel[] vokabeln = new Vokabel[20];
5. int freierIndex = 0;
6. int aktuell = 0;
7. // Ende Attribute
8. public Vokabeltrainer() {
9. // Frame-Initialisierung
10. super();
11. vokabeln[0] = new Vokabel("Tisch", "table");
12. vokabeln[1] = new Vokabel("Stuhl", "chair");
13. vokabeln[2] = new Vokabel("Tür", "door");
14. vokabeln[3] = new Vokabel("Apfel", "apple");
15. vokabeln[4] = new Vokabel("Eichhörnchen", "squirrel");
16. vokabeln[5] = new Vokabel("Informatik", "computer science");
17. freierIndex  = 6;
18. […]
19. } // end of public Vokabeltrainer
20. // Anfang Methoden
21. […]
22. public void bZeigedeutschesWort\_ActionPerformed(ActionEvent evt) {
23. aktuell = (int)(Math.random()\*freierIndex);
24. jTextFieldDeutsch.setText(vokabeln[aktuell].getWortD());
25. jTextFieldEnglisch.setText("");
26. }
27. public void bPruefen\_ActionPerformed(ActionEvent evt) {
28. String wd = jTextFieldDeutsch.getText();
29. String we = jTextFieldEnglisch.getText();
30. boolean richtig = vokabeln[aktuell].pruefen(wd, we);
31. if(richtig) jLabelAusgabe.setText("Das war richtig, super");
32. else jLabelAusgabe.setText("Leider falsch");
33. }
34. public void bHinzufuegen\_ActionPerformed(ActionEvent evt) {
35. if(freierIndex < vokabeln.length){
36. String wd = jTextFieldDeutsch.getText();
37. String we = jTextFieldEnglisch.getText();
38. vokabeln[freierIndex] = new Vokabel(wd, we);
39. freierIndex++;
40. }
41. }
42. } // end of class Vokabeltrainer

Beispiel 2: Verwendung der Klasse Vokabel

In Zeile 4 wird eine Reihung mit Namen vokabeln vom Typ Vokabel der Länge 20 angelegt. Das heißt, die Reihung vokabeln kann 20 Objekte des Typs Vokabel aufnehmen. Die ersten sechs Plätze der Reihung vokabeln werden beim Starten des Programms bereits mit Einträgen belegt. Dazu werden in den Zeilen 12 bis 17 mithilfe des Konstruktors der Klasse Vokabel sechs exempla­rische Objekte vom Typ Vokabel erzeugt und den ersten sechs Plätzen der Reihung vokabeln zugewiesen. In der globalen Variablen freierIndex ist der Index des ersten Platzes in der Reihung vokabeln gespeichert, der noch nicht mit einem Objekt der Klasse Vokabel belegt ist.

Das Vokabeltrainer-Programm verfügt über verschiedene Buttons, mit denen bestimmte Aktionen ausgelöst werden können, die in den entsprechenden Methoden ab Zeile 24 definiert sind. Die Methode, die zu dem Button mit der Beschriftung „zeige deutsches Wort“ gehört, ist in den Zeilen 24 bis 28 definiert. In den Grenzen von 0 bis freierIndex wird eine Zufallszahl erzeugt und in der globalen Variablen aktuell gespeichert. Die deutsche Bezeichnung der Vokabel an dieser zufälligen Position in der Reihung vokabeln wird in dem Textfeld für die deutsche Bezeichnung ausgegeben. Das Auslesen der deutschen Bezeichnung aus dem Objekt vom Typ Vokabel erfolgt mithilfe der Operation getWortD. Der Aufruf der Operation erfolgt nach dem Muster Objekt.Methodenname(), hier also vokabeln[aktuell].getWortD(). Der Inhalt des Textfeldes für die englische Bezeichnung wird gelöscht. Zeile 26 kann auch in mehrere Anweisungen zerlegt werden, indem die Zwischenergebnisse in temporären Variablen gespeichert werden:

1. Vokabel vtemp = vokabeln[aktuell];
2. String dBezeichnung = vtemp.getWortD();
3. jTextFieldDeutsch.setText(dBezeichnung);

Der Anwender bzw. die Anwenderin kann nun die fehlende englische Bezeichnung eintragen und anschließend den Button mit der Beschriftung „prüfen“ anklicken.

In den Zeilen 30 bis 36 ist die Aktion für den Button mit der Beschriftung „prüfen“ definiert. Die Einträge aus den Textfeldern für die deutsche und die englische Bezeichnung werden ausgelesen. In der globalen Variablen aktuell ist noch der Index der Reihung vokabeln gespeichert, der zu der Vokabel gehört, die gerade abgefragt wird. Für dieses Objekt wird nun die Operation pruefen aufgerufen. Als Parameter erhält sie die zuvor aus den Textfeldern ausgelesenen Zeichenketten. Je nachdem, ob die Methode den Wert true oder false zurückgibt, erhält der Anwender bzw. die Anwenderin eine entsprechende Rückmeldung.

Mithilfe des Buttons „hinzufügen“ kann eine neue Vokabel zu der Reihung vokabeln hinzugefügt werden, wenn noch ein Platz frei ist. In den Zeilen 38 bis 45 ist entsprechend definiert, dass an der nächsten freien Position ein neues Objekt vom Typ Vokabel eingefügt wird, das mithilfe des Konstruktors für die aktuellen Werte in den Textfeldern für die deutsche und die englische Bezeichnung erzeugt wird. Die globale Variable freierIndex wird entsprechend angepasst.



Abbildung : Exemplarische Benutzeroberfläche zur Klasse Vokabeltrainer in Beispiel 2.

**Aufgabe 1:**

1. In Beispiel 2 werden an verschiedenen Stellen Operationen (Methoden) eines Objekts ausgeführt. Geben Sie Beispiele an und markieren Sie das Objekt blau und den Methodennamen orange.
2. Markieren Sie alle Stellen grün, an denen ein neues Objekt vom Typ Vokabel erzeugt wird.

**Aufgabe 2:**

1. Nehmen Sie folgende Erweiterungen an dem Programm Vokabeltrainer vor:
2. Beim Start des Programms soll an Position 6 der Reihung vokabeln ein weiterer Eintrag für die Vokabel „Löwe“ - „lion“ erfolgen.
3. Ergänzen Sie eine globale Variable vokabelDesTages vom Typ Vokabel. Speichern Sie in der Variablen eine Vokabel Ihrer Wahl. Diese soll beim Starten des Programms angezeigt werden. Nehmen Sie das Objekt vokabelDesTages auch mit in die Reihung vokabeln auf.
4. Implementieren Sie die Operation, die ausgeführt wird, wenn der Button mit der Beschrif­tung „zeige englisches Wort“ angeklickt wird, analog zu der Operation, die ausgeführt wird, wenn der Button mit der Beschriftung „zeige deutsches Wort“ angeklickt wird.
5. Verändern Sie die Operation zum Anzeigen eines deutschen bzw. englischen Wortes so, dass ein Wort angezeigt wird, das bislang noch nicht richtig eingegeben wurde, d. h. bei dem das Attribut gewusst den Wert false hat. Entscheiden Sie selbst, wie verfahren werden soll, wenn bereits alle Vokabeln richtig eingegeben wurden, also das Attribut gewusst bei allen Vokabeln den Wert true hat.
6. Verändern Sie die Rückmeldung an den Anwender bzw. die Anwenderin so, dass beim Überprüfen für die aktuelle Vokabel die Anzahl der bislang schon erfolgten Versuche mit ausgegeben wird.
7. Ergänzen Sie einen Button, mit dem bei allen Vokabeln der Wert des Attributs gewusst auf false und der Wert des Attributs versuche auf 0 gesetzt werden kann.
8. Erweitern Sie die Klasse Vokabel um einen Konstruktor, der zusätzlich zu den Werten für die deutsche und die englische Bezeichnung einen Wahrheitswert für das Attribut gewusst als Parameter erhält. Testen Sie Ihren Konstruktor, indem sie damit entsprechende Objekte in der Reihung vokabeln einfügen.
9. Ergänzen Sie in der Klasse Vokabel ein Attribut kategorie, in dem gespeichert werden kann, ob es sich um ein Nomen, ein Verb, ein Adjektiv oder Sonstiges handelt.

Beschreiben Sie verschiedene Möglichkeiten, wie dieses Attribut in dem Programm Vokabeltrainer genutzt werden kann.

## Sichtbarkeit von Attributen und Operationen

In der Klasse Vokabel wurden nach dem Prinzip der Kapselung alle Attribute mit dem Schlüsselwort private und alle Operationen mit dem Schlüsselwort public versehen. Grundsätzlich können beide Schlüsselwörter sowohl für Attribute als auch für Operationen verwendet werden. Man spricht hier von der Sichtbarkeit der Attribute bzw. Operationen. Attribute und Operationen, die das Schlüsselwort public erhalten, sind außerhalb einer Klasse bzw. eines Objekts sichtbar. Der Zugriff von außen kann mithilfe der Notation Objekt.Attributname bzw. Objekt.Methodenname()erfolgen. Man spricht von **öffentlichen** Attributen bzw. Operationen. Attribute und Operationen, die das Schlüsselwort private erhalten, sind außerhalb der Klasse bzw. eines Objektes nicht sichtbar. Auf **private** Attribute kann somit von außen nicht direkt zugegriffen werden, sondern nur über entsprechende get- und set-Methoden, die die Klasse ggf. zur Verfügung stellt. Private Operationen sind Hilfsoperationen, die nur von anderen Operationen innerhalb der gleichen Klasse aufgerufen werden können. Sie sind nicht Teil der Schnittstelle eines Objekts nach außen. In einer Klassenkarte wird die Sichtbarkeit durch ein Pluszeichen für öffentlich und ein Minuszeichen für privat gekennzeichnet (s. Abbildung 1).

**Aufgabe 3:** Es soll eine Klasse Person mit den Attributen vorname, nachname, wohnort, bundesland, alter, istErwachsen erstellt werden. Entscheiden Sie für die Attribute jeweils begründet, ob es problematisch wäre, wenn die Attribute das Schlüsselwort public erhalten und damit direkt von außen zugänglich und veränderbar sind.

## Exkurs: Globale und lokale Variablen

Variablen, die in einer Klasse außerhalb der Methoden definiert werden, nennt man **globale Varia­blen**. Die Attribute einer Klasse sind also globale Variablen. Innerhalb der Klasse kann man in jeder Methode darauf zugreifen, unabhängig davon, ob sie als private oder public deklariert wurden.

Zusätzlich gibt es häufig noch Parameter oder Hilfsvariablen, die in einer Methode oder hier sogar erst innerhalb einer Schleife oder Verzweigung definiert werden. Diese Variablen stehen auch nur in dem entsprechenden durch geschweifte Klammern eingeschlossenen Block zur Verfügung. Nach dem Beenden der Methode bzw. Schleife oder Verzweigung wird die Variable automatisch wieder vernichtet. Ein Zugriff ist nicht mehr möglich. Man spricht hier von **lokalen Variablen**.

Insbesondere bei Konstruktoren möchte man manchmal, dass Parameter genauso heißen wie die globalen Variablen, für die hier ein entsprechender Wert übergeben wird. In diesem Fall kann durch Voranstellen von this. vor den Variablennamen das eigene Objekt referenziert und so auf eine globale Variable zugegriffen werden, für die eine gleichnamige lokale Variable existiert. Wird nur der Variablenname ohne this. verwendet, ist immer die lokale Variable gemeint. Beispiel 3 zeigt den Anfang der Definition der Klasse Vokabel mit einem Konstruktor, bei dem die Parameter genauso heißen wie die entsprechenden globalen Variablen. In Zeile 11 bezieht sich this.wortD auf die globale Variable wortD und nur wortD auf den Parameter wortD.

1. public class Vokabel{
2. // Anfang Attribute
3. private String wortD;
4. private String wortE;
5. private int versuche;
6. private boolean gewusst;
7. // Ende Attribute
8. //Konstruktor
9. public Vokabel(String wortD, String wortE){
10. **this.**wortD = wortD;
11. **this.**wortE = wortE;
12. versuche = 0;
13. gewusst = false;
14. }
15. …}

Beispiel 3: Globale Variablen und Verwendung gleichnamiger Parameter im Konstruktor

## Objekte als Referenzvariablen

Im Unterschied zu Variablen mit einem primitiven Datentyp (z. B. int, boolean) sind Objekte

Referenzvariablen. Beim Erzeugen eines Objekts wird der Variablen eine Adresse im Speicher zugewiesen. An dieser Adresse wird jedoch nicht der Inhalt des Objekts, sondern ein Verweis (Refe­renz) auf eine andere Stelle im Speicher abgelegt, an der dann der Inhalt des Objekts zu finden ist. Die **Referenz** auf den Inhalt des Objekts ändert sich daher nicht, wenn einzelne Werte der Attribute eines Objekts verändert werden. Die Referenz identifiziert ein Objekt somit eindeutig und unab­hängig von den konkreten Werten der Attribute. Sollen zwei Objekte verglichen werden, ist daher zu unterscheiden, ob die Identität, also die Referenz der Objekte, oder ihr Inhalt, also die Belegung der Attribute, verglichen werden soll. Der Vergleichsoperator == gibt nur dann den Wert wahr zurück, wenn beide Variablen denselben Bereich im Speicher referenzieren, es sich also tat­säch­lich um dasselbe Objekt handelt. Sollen Objektvariablen hingegen als gleich gelten, wenn die Werte ihrer Attribute gleich sind, muss eine entsprechende Methode equals in der Klasse definiert werden, die die Werte der Attribute mit den Werten der Attribute eines als Parameter übergebenen Objekts vergleicht. Diese kann dann für den Vergleich eines Objekts mit einem anderen verwendet werden.

**Aufgabe 4:**

1. Erläutern Sie die Abbildungen 4 bis 6.
2. Definieren Sie für die Klasse Vokabel eine Operation equals, die den Inhalt, d.h. die Werte der Attribute mit einem zweiten Objekt vom Typ Vokabel vergleicht. Die Operation erhält das zweite Objekt vom Typ Vokabel als Parameter. Die Operation gibt genau dann den Wert wahr (true) zurück, wenn das übergebene Objekt für jedes Attribut den gleichen Wert hat wie das Objekt, für das die Methode aufgerufen wird.
3. Entscheiden Sie für die Situationen, die in den Abbildungen 4 bis 6 dargestellt sind, jeweils, ob die beiden folgenden Bedingungen erfüllt sind:
4. v1 == v2
5. v1.equals(v2)

Vokabel v1 = new Vokabel("Tisch", "table");

wortD = "Tisch"

wortE = "table"

versuche = 0

gewusst = falsch

wortD = "Stuhl"

wortE = "chair"

versuche = 0

gewusst = falsch

640

981

Vokabel v2 = new Vokabel("Stuhl", "chair");

|  |  |
| --- | --- |
| Adresse | Inhalt |
| … |  |
| 123 | 640 |
| 124 |  |
| 125 | 981 |
| 126 |  |
| … |  |

Abbildung : Veranschaulichung der Referenzvariablen v1 und v2 nach dem Erzeugen

Vokabel v1 = new Vokabel("Tisch", "table");

wortD = "Tisch"

wortE = "table"

versuche = 0

gewusst = falsch

wortD = "Tisch"

wortE = "table"

versuche = 0

gewusst = falsch

640

981

Vokabel v2 = new Vokabel("Stuhl", "chair");

v2.setWortD("Tisch");

v2.setWortE("table");

|  |  |
| --- | --- |
| Adresse | Inhalt |
| … |  |
| 123 | 640 |
| 124 |  |
| 125 | 981 |
| 126 |  |
| … |  |

Abbildung : Veranschaulichung der Belegung des Speichers nach Erzeugen der Varibalen v1 und v2 und Ändern der Werte von v2

Vokabel v1 = new Vokabel("Tisch", "table");

wortD = "Tür"

wortE = "door"

versuche = 0

gewusst = falsch

640

Vokabel v2 = v1;

v1.setWortD("Tür");

v2.setWortE("door");

|  |  |
| --- | --- |
| Adresse | Inhalt |
| … |  |
| 123 | 640 |
| 124 |  |
| 125 | 640 |
| 126 |  |
| … |  |

Abbildung : Veranschaulichung der Belegung des Speichers nach Erzeugen der Variablen v1 und v2 und Ändern der Werte

**Aufgabe 5:** Skizzieren Sie in der beiliegenden Vorlage für den folgenden Ausschnitt eines Programms den Inhalt der Reihung meineVokabeln an den markierten Stellen. Färben Sie identische Objekte in der gleichen Farbe. Abbildung 7 zeigt den Inhalt der Reihung in Zeile 5.

1. Vokabel[] meineVokabeln = **new** Vokabel[3];
2. meineVokabeln[0] = **new** Vokabel("Apfel", "apple");
3. meineVokabeln[1] = **new** Vokabel("Eule", "owl");
4. meineVokabeln[2] = **new** Vokabel("Auto", "car");
5. //Speicherbelegung skizzieren
6. meineVokabeln[1].setWortD(meineVokabeln[0].getWortD());
7. meineVokabeln[1].setWortE(meineVokabeln[0].getWortE());
8. //Speicherbelegung skizzieren
9. meineVokabeln[0] = meineVokabeln[2];
10. //Speicherbelegung skizzieren
11. meineVokabeln[2].setWortE("automobile");
12. //Speicherbelegung skizzieren

**Zeile 5:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | wortD = "Apfel"  wortE = "apple"  versuche = 0  gewusst = falsch | wortD = "Eule"  wortE = "owl"  versuche = 0  gewusst = falsch | wortD = "Auto"  wortE = "car"  versuche = 0  gewusst = falsch |
| meineVokabeln | 0 | 1 | 2 |

Abbildung : Belegung der Reihung meineVokabeln in Zeile 5

## Aufgaben zum Erstellen und Verwenden von Klassen

**Aufgabe 6:** In dieser Aufgabe soll ein kleines Programm entstehen, mit dem die Geburtstage der Familie, Freundinnen und Freunde usw. verwaltet werden können.

1. Legen Sie dazu zunächst eine Klasse Geburtstag an, in der die Attribute name, tag, monat und geburtsjahr gespeichert werden können. Neben einem Konstruktor und den benötigten get- und set-Methoden soll auch eine Operation, die das aktuelle Alter der Person berechnet, enthalten sein.

**Tipp:** Importieren Sie das Paket java.time.\*

Mithilfe der folgenden Methodenaufrufe erhalten Sie den aktuellen Tag, den aktuellen Monat bzw. das aktuelle Jahr des heutigen Tages als Ganzzahl.

ZonedDateTime.now().getDayOfMonth()

ZonedDateTime.now().getMonthValue()

ZonedDateTime.now().getYear()

1. Erstellen Sie ein Programm ähnlich wie in Abbildung 8, das für einige Geburtstage, die in einer globalen Reihung vom Typ Geburtstag vorliegen, die folgenden Funktionen bietet:

* Ausgabe des Geburtstags und des aktuellen Alters zu einem Namen.
* Hinzufügen eines Geburtstags.
* Ausgabe aller Geburtstagskinder in einem bestimmten Monat.
* Anzeige, ob heute jemand Geburtstag hat.

Ergänzen Sie weitere Funktionen nach Ihren Vorstellungen.

**Tipp:**

* Gehen Sie vereinfachend davon aus, dass Namen nicht doppelt auftreten.
* Ergänzen Sie bei Bedarf weitere Operationen in der Klasse Geburtstag.



Abbildung : Mögliche Oberfläche für das Programm in Aufgabe 6

**Aufgabe 7:** Eine Klasse DVD hat die privaten Attribute titel vom Typ Zeichenkette und wieOftGeguckt vom Typ Ganzahl. Bei der Erstellung eines Objekts der Klasse DVD soll man den Titel angeben können. Der Wert des Attributs wieOftGeguckt wird zu Beginn auf 0 gesetzt. Die Klasse DVD soll außerdem über eine Operation filmGucken verfügen, die das Attribut wieOftGeguckt um eins erhöht. Außerdem benötigt die Klasse für jedes Attribut eine get-Methode.

Eine weitere Klasse DVDRegal soll eine bestimmte Anzahl an DVDs aufnehmen können. Die Anzahl wird dem Konstruktor als Ganzzahl übergeben. Man befüllt das Regal mit einer Operation add, die als Parameter den Titel einer DVD als Zeichenkette erhält und ein entsprechendes Objekt vom Typ DVD erstellt und aufnimmt, wenn noch Platz im Regal ist. Außerdem gibt es die Operationen alleGucken und filmGucken(titel: Zeichenkette), mit der alle DVDs bzw. die über den Parameter titel übergebene DVD geguckt werden können. Das Schauen einer DVD wird durch den Aufruf der Operation filmGucken der Klasse DVD simuliert. Gehen Sie vereinfachend davon aus, dass zwei DVDs nicht den gleichen Titel haben.

Implementieren Sie die Klassen DVD und DVDRegal. Erstellen Sie auch ein geeignetes Programm zum Testen der Funktionalität der Klassen, z. B. wie in Abbildung 9. Ergänzen Sie bei Bedarf weitere Operationen in den Klassen, die Sie für Ihr Testprogramm benötigen.



Abbildung : Exemplarische Oberfläche für ein Programm zum Testen der Klassen DVD und DVDRegal

## Klassendiagramme

Klassendiagramme haben wir bislang nur in Form einer Klassenkarte für eine Übersicht über eine einzelne Klasse verwendet. **Klassendiagramme** können aber auch die Beziehungen verschiedener Klassen zueinander veranschaulichen. In Aufgabe 7 verwaltet eine Klasse DVDRegal mehrere Objekte des Typs DVD. Ein Objekt des Typs DVDRegal ist wiederum in der Klasse zum Testen enthalten. Diese Beziehungen stellt das Klassendiagramm in Abbildung 10 dar. Zu den Klassen sind nur die im Kontext der Aufgabenstellung zentralen Attribute und Operationen angegeben. Durch „…“ wird jeweils angezeigt, dass es in den Klassen noch weitere Attribute bzw. Operationen gibt.



Abbildung : Klassendiagramm zur DVD-Verwaltung aus Aufgabe 7

**Aufgabe 8:** Das Einstiegsbeispiel Vokabeltrainer könnte wie folgt erweitert werden: Eine Klasse Lektion verwaltet alle Vokabeln einer Lektion in einer eindimensionalen Reihung vom Typ Vokabel. Das Hauptprogramm Vokabeltrainer verfügt für jeden Schuljahrgang von 5 bis 10 jeweils über eine eindimensionale Reihung vom Typ Lektion, in der alle Lektionen des jeweiligen Jahrgangs enthalten sind.

Erstellen Sie ein entsprechendes Klassendiagramm. Ergänzen Sie in den Klassen geeignete Attribute, Konstruktoren oder Operationen.

## Weitere Konzepte der objektorientierten Programmierung

Die im Weiteren beschriebenen Konzepte lassen sich gut an Klassen für geometrische Objekte veranschaulichen, die auch grafisch gezeichnet werden. Die Umsetzung erfolgt daher in Processing[[2]](#footnote-2), da grafische Ausgaben hier besonders einfach erzeugt werden können. Die Konzepte sind aber allgemeingültig für die objektorientiere Programmierung. Die in den Beispielen und Aufgaben verwendeten Klassen können auch mithilfe des JavaEditors erstellt und eine entsprechende grafische Ausgabe in einem *JFrame* erzeugt werden.

Mithilfe von Zeichenbefehlen lassen sich verschiedene Figuren in das Anwendungsfenster zeichnen. Das Zeichnen erfolgt durch Einfärben der entsprechenden Pixel. Soll die Zeichnung später verändert werden, z. B. die Farbe oder Position einer Figur, muss diese mit veränderten Parametern neu gezeichnet werden. Dazu muss im Programm gespeichert werden, an welcher Position welche Figur bereits gezeichnet wurde. Um alle Parameter einer Figur zusammenzufassen, bietet es sich an, eine entsprechende Klasse anzulegen. Wir schauen uns in Aufgabe 9 zunächst ein Beispiel an, bei dem als Reaktion auf einen Mausklick ein Rechteck an die Position der Maus gezeichnet wird. Jedes Rechteck soll als Objekt einer Klasse Rechteck in einer Reihung vom Typ Rechteck gespeichert werden. Dadurch ist es möglich im Hauptprogramm zusätzliche Funktionen zur Verfügung zu stellen, z. B. das Ändern der Farbe aller Rechtecke auf Tastendruck.

**Aufgabe 9:**

1. Untersuchen Sie den Aufbau des Programms RechteckeZeichnen und der Klasse Rechteck.
2. Stellen Sie den Aufbau der Klasse Rechteck in einer Klassenkarte dar. Bei den get- und set-Operationen reicht es, wenn Sie diese exemplarisch zu einem Attribut angeben und die restlichen durch „…“ andeuten.
3. Ergänzen Sie das Programm um weitere Funktionen zum Zeichnen und Verändern der Rechtecke.

**Aufgabe 10:** Das beiliegende Programm Blumenwiese zeichnet 10 Blumen an zufälligen Positionen und bewegt sie anschließend hin und her. Die Parameter der Blumen werden aktuell in verschiedenen Reihungen im Programm abgelegt.

1. Erstellen Sie eine Klasse Blume, in der alle benötigten Eigenschaften einer Blume gespeichert werden können. Die Klasse Blume soll außerdem eine Operation zeichnen und eine Operation bewegen zur Verfügung stellen.
2. Verändern Sie das Programm so, dass es alle Blumen in einer eindimensionalen, globalen Reihung vom Typ Blume verwaltet. Außerdem soll das Programm die Operationen zeichnen und bewegen verwenden.

**Aufgabe 11:** Das Programm WindmuehlenAnimation zeichnet 15 Windmühlen an zufälligen Positionen. Die Windmühlen drehen sich anschließend im Wind. Die Parameter der Windmühlen werden aktuell in einer zweidimensionalen Reihung im Programm abgelegt.

1. Erstellen Sie eine Klasse Windmuehle, in der alle benötigten Eigenschaften einer Windmühle gespeichert werden können. Die Klasse Windmuehle soll außerdem eine Operation zeichnen und eine Operation drehen zur Verfügung stellen.
2. Verändern Sie das Programm so, dass es alle Windmühlen in einer global definierten eindimensionalen Reihung vom Typ Windmuehle verwaltet. Außerdem soll das Programm die Operationen zeichnen und drehen verwenden.
3. Verändern Sie das Programm nach Ihren Vorstellungen, z. B. um die Möglichkeit von Nutzereingaben.

## Das Konzept der Vererbung

Das Programm RechteckeZeichnen aus Aufgabe 9 ermöglicht bislang nur das Zeichnen von Rechtecken. Das Programm soll nun so erweitert werden, dass der Anwender bzw. die Anwenderin wählen kann, ob er bzw. sie ein Rechteck, ein Quadrat, einen Kreis oder eine Ellipse zeichnen möchte. Für jede der vier Figuren wird daher eine Klasse benötigt, um jede Figur entsprechend als Objekt speichern zu können.

**Aufgabe 12:**

1. Entwerfen Sie für eine Klasse Rechteck, Quadrat, Kreis und Ellipse jeweils eine Klassenkarte.
2. Untersuchen Sie, welche Gemeinsamkeiten und welche Unterschiede die Klassen haben.

Um Eigenschaften und Operationen, die alle Figuren gemeinsam haben, nicht jedes Mal neu implementieren zu müssen, kann das Konzept der **Vererbung** verwendet werden. Wir erstellen zunächst eine **Oberklasse** Figur, die alle Attribute enthält, die jede Klasse später benötigt. Von dieser Oberklasse können wir dann z. B. eine **Unterklasse** Rechteck ableiten. Die Klasse Rechteck erbt automatisch alle Attribute und Operationen der Oberklasse Figur, ohne dass wir sie noch einmal explizit angeben. In der Klasse Rechteck müssen nur die zusätzlichen Attribute und Operationen sowie ein Konstruktor festgelegt werden. Ein Quadrat ist ein spezielles Rechteck, bei dem die Attribute breite und hoehe den gleichen Wert haben. Die Klasse Quadrat kann daher von der Klasse Rechteck abgeleitet werden. In diesem Fall benötigt die Klasse Quadrat keine zusätzlichen Attribute. Es müssen lediglich die Operationen zum Setzen der Höhe bzw. Breite angepasst werden, damit sichergestellt ist, dass beide Attribute immer den gleichen Wert haben. Abbildung 11 stellt die Zusammenhänge in Form eines Klassendiagramms dar. Die Vererbung wird in einem Klassendiagramm durch einen Pfeil mit dreieckiger, nicht ausgefüllter Pfeilspitze dargestellt, der von der Unterklasse zur Oberklasse zeigt. In der Unterklasse werden nur die zusätzlichen Attribute und die Operationen, die neu hinzukommen oder verändert werden, aufgeführt. Eine genauere Erläuterung zu Operationen, die in einer Unterklasse anders definiert werden als in der ****Oberklasse, folgt im Abschnitt *Konzept der Polymorphie*.

Abbildung : Vererbung am Beispiel geometrischer Figuren

**Aufgabe 13:** Ergänzen Sie in dem Klassendiagramm in Abbildung 11 eine Klasse Ellipse und eine Klasse Kreis, die jeweils von einer Oberklasse abgeleitet werden.

### Hinweise zur Implementierung

Bei der Implementierung wird die Oberklasse, von der eine neue Klasse abgeleitet wird, mithilfe des Schlüsselwortes extends angegeben.

**Beispiel**: public class Rechteck extends Figur{…}

Da Konstruktoren nicht vererbt werden, muss im Konstruktor der Unterklasse zunächst ein Konstruktor der Oberklasse aufgerufen werden. Dies kann explizit mit der Anweisung super(); geschehen. Hier können ggf. auch Parameter ergänzt werden. Ist die erste Anweisung im Konstruktor der Unterklasse kein Konstruktor der Oberklasse, wird automatisch der Standardkonstruktor der Oberklasse ausgeführt.

Zu beachten ist bei der Vererbung die Sichtbarkeit der Attribute. Wird ein Attribut in der Oberklasse als private deklariert, ist es selbst in den Unterklassen nicht sichtbar. Es kann hier also nur über die get- und set-Methoden der Oberklasse abgefragt bzw. verändert werden. Wird ein Attribut hingegen als public deklariert, ist es von außen sichtbar, also auch in der Unterklasse. Um die Implementierung der Unterklassen zu vereinfachen, können die Attribute einer Oberklasse als public deklariert werden. Um im Sinne der Kapselung den Zugriff auf die Attribute von einer anderen Klasse aus, die nicht Teil der Vererbungshierarchie ist, wie gewohnt über Operationen zu realisieren, können zusätzlich get- und set-Methoden angelegt werden.

## Das Konzept der Polymorphie

### Überschreiben von Operationen

Ein weiterer Vorteil des Konzepts der Vererbung ist, dass alle Unterklassen auch vom Typ der Oberklasse sind. Dadurch können in einer Reihung vom Typ Figur sowohl Objekte des Typs Quadrat als auch des Typs Rechteck, Kreis oder Ellipse abgelegt werden.

In diesem Zusammenhang spielt das **Überschreiben** von Operationen eine Rolle. Damit für ein Objekt des Typs Figur die Operation zeichnen ausgeführt werden kann, muss diese auch in der Oberklasse Figur vorhanden sein, auch wenn Sie hier noch nicht allgemein definiert werden kann. Der Inhalt der Methode kann zunächst leer bleiben oder eine Standard-Figur zeichnen. Wird in der Unterklasse ebenfalls eine Operation zeichnen mit gleichem Rückgabetyp und gleichen Parametern definiert, so überschreibt diese Operation die gleichnamige Operation der Oberklasse. Das heißt, wenn ein Objekt vom Typ Rechteck in einer Variablen vom Typ Figur abgelegt wird und anschließend die Operation zeichnen für dieses Objekt aufgerufen wird, so wird die Operation so ausgeführt, wie sie in der Unterklasse Rechteck definiert ist. Ein weiteres Beispiel für das Überschreiben von Operationen sind die set-Methoden der Klasse Quadrat. Da hier anders als in der Klasse Rechteck der übergebene Parameter sowohl dem Attribut breite als auch dem Attribut hoehe zugewiesen werden muss, werden die Operationen setBreite und setHoehe in der Klasse Quadrat neu definiert und damit die entsprechenden Methoden der Klasse Rechteck überschrieben.

Manchmal soll die Definition einer Operation in der Unterklasse die Definition in der Oberklasse nur ergänzen. Mit einer Anweisung nach dem Muster super.Methodenname(Parameter); kann daher in der Unterklasse die Definition der Oberklasse für eine Operation aufgerufen werden. Die Operation setBreite der Klasse Quadrat könnte daher wie folgt implementiert werden:

1. public void setBreite(int b){
2. super.setBreite(b);
3. super.setHoehe(b);
4. }

**Aufgabe 14:**

1. Implementieren Sie die Klassen Rechteck, Quadrat, Ellipse und Kreis als abgeleitete Klassen der Klasse Figur.
2. Verändern Sie das Programm RechteckeZeichnen zu einem Programm FigurenZeichnen, welches das Zeichnen verschiedener Figuren ermöglicht. Alle Figuren sollen in einer global definierten eindimensionalen Reihung vom Typ Figur verwaltet werden.

**Hinweis**: Als Hilfestellung können Sie das unvollständige Programm FigurenZeichnen als Vorlage verwenden.

**Aufgabe 15:** In dieser Aufgabe soll das Programm Blumenwiese aus Aufgabe 10 erweitert werden.

1. Leiten Sie von der Klasse Blume eine Klasse BlumeMitBlaettern ab, welche eine Blume mit zwei zusätzlichen Blättern zeichnet.
2. Verändern Sie das Programm so, dass sowohl Blumen mit als auch ohne Blätter gezeichnet werden.
3. Erläutern sie an diesem Beispiel das Konzept des Überschreibens.
4. Untersuchen Sie, ob es hier alternativ zu der Verwendung einer Unterklasse andere geeignete Modellierungen gibt.

### Überladen von Operationen

Neben dem Überschreiben von Operationen kann es manchmal sinnvoll sein, Operationen innerhalb einer Klasse mit verschiedenen Parametern anzubieten. In diesem Fall spricht man von **Überladen**. In den vorangegangenen Beispielen gab es häufig Konstruktoren mit unterschiedlichen Parametern. Der Standardkonstruktor erhält gar keine Parameter. Diesen stellt eine Klasse immer zur Verfügung, auch wenn er nicht explizit implementiert wird. Zusätzlich kann es weitere Konstruktoren geben, die Parameter für die Belegung aller oder einen Teil der Attribute erhalten. Die übrigen Attribute werden ggf. mit Standardwerten belegt.

Sowohl das Konzept des Überscheibens als auch des Überladens bezeichnet man in der objektorientierten Programmierung als **Polymorphie** (Vielgestaltigkeit).

**Aufgabe 16:** Geben Sie anhand der Klassen, die Sie in den Aufgaben 11 und 14 verwendet haben, Beispiele für Konstruktoren an, die das Konzept des Überladens umsetzen.

**Aufgabe 17:** Ergänzen Sie in der Klasse Windmuehle aus Aufgabe 11 eine weitere Operation drehen, die einen Parameter erhält, der festlegt, um wie viel Grad die Windmühle gedreht werden soll. Verändern Sie das Programm WindmuehlenAnimation geeignet, um die Operation drehen mit und ohne Parameter zu testen.

**Aufgabe 18:**

1. Ergänzen Sie in dem Programm aus Aufgabe 9 in der Klasse Rechteck verschiedene Definitionen der Operation zeichnen, die unterschiedliche Parameter verwenden. Die Operation zeichnen kann beispielsweise Parameter für die Füllfarbe oder für die Position erhalten.
2. Für die Klasse Rechteck soll es die zwei folgenden Operationen geben:
3. zeichnen(x: Ganzzahl, y: Ganzzahl)
4. zeichnen (breite: Ganzzahl, hoehe: Ganzzahl)

Während die erste Operation das Rechteck an einer neuen Position zeichnen soll, soll die zweite Operation das Rechteck mit einer neuen Breite und einer neuen Höhe zeichnen.

Stellen Sie eine Vermutung auf, warum in der Form nicht beide Operationen gleichzeitig in der Klasse Rechteck implementiert werden können. Welche alternative Lösung gibt es hier?

1. Ergänzen Sie in der Klasse Rechteck eine Operation flaecheBerechnen, welche die Fläche des Rechtecks berechnet und als Fließkommazahl zurückgibt.
2. Begründen Sie, dass es nicht möglich ist, eine weitere Operation flaecheBerechnen zu ergänzen, welche ebenfalls die Fläche des Rechtecks berechnet, aber diese als Ganzzahl oder Zeichenkette zurückgibt.

Dieses Werk ist lizenziert unter einer [Creative Commons Namensnennung - Nicht-kommerziell - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz](http://creativecommons.org/licenses/by-nc-sa/4.0/). Von der Lizenz ausgenommen ist das InfSII-Logo.

Für die korrekte Ausführbarkeit der beiliegenden Quelltexte wird keine Garantie übernommen. Auch für Folgeschäden, die sich aus der Anwendung der Quelltexte oder durch eventuelle fehlerhafte Angaben ergeben, wird keine Haftung oder juristische Verantwortung übernommen.

1. Der Java-Quellcode für *JFrames* in diesem Leitfaden wurde mithilfe des Java Editors von Gerhard Röhner erstellt: <https://javaeditor.org/> [Datum des Zugriffs: 01.02.2023] [↑](#footnote-ref-1)
2. Die Programmierumgebung Processing wurde 2001 von Ben Fry und Casey Reas initiiert. Nähere Informationen finden Sie unter <https://processing.org/> [↑](#footnote-ref-2)