Rückblick, Polymorphie

3

Einstieg

In der neunten Klasse wurde die objektorientierte Modellierung und Programmierung eingeführt. In diesem Kapitel werden die zentralen Konzepte wiederholt und vertieft. Dabei wird insbesondere erklärt, was Objekte und Variablen gemeinsam haben.

- Beschreibt den Unterschied zwischen Klassen und Objekten.
- Nennt mindestens drei verschiedene Kontrollstrukturen.
- Nennt drei verschiedene Diagrammarten und gebt jeweils an, wofür man das Diagramm verwendet.

Am Ende dieses Kapitels hast du gelernt, …

- warum Klassen als Datentypen interpretiert werden können.
- welchen Datentyp ein Objekt genau hat und haben kann.
- was Polymorphie ist und wie sie sich auf Methodenaufrufe auswirkt.

EINSTIEG

August hat die Bewegungsgleichungen im Physikunterricht gelernt und möchte nun eine Simulation hierzu programmieren.

- Beschreibe, welche Kenntnisse und Hilfsmittel August für sein Vorhaben besitzen muss.
- Überlege, welche Schritte notwendig sind, um die Simulation zu programmieren.

ERARBEITUNG

Überblick über die Programmierung

Bei der Programmierung wird üblicherweise eine Entwicklungsumgebung verwendet. Diese übersetzt den Programmtext in Maschinensprache und führt ihn aus. Tritt beim Übersetzen ein Fehler auf, liegt ein Syntaxfehler vor. Tritt der Fehler beim Ausführen auf, so handelt es sich um einen Semantikfehler.

A1 Fehlerquellen nennen

a) Nenne drei mögliche Arten von Syntaxfehlern mit Beispielen.
b) Nenne drei mögliche Ursachen für Semantikfehler mit je einem Beispiel.
c) Begründe, ob ein Programmtext mit Syntaxfehlern semantisch sinnvoll sein kann.

Programmierung in Greenfoot mit Java

In ➥ Fit für die Praxis (VI.) findest du eine Übersicht über die Syntax von Anweisungen und Kontrollstrukturen in Java.

In Greenfoot kann man mit der Programmiersprache Java programmieren. In Java existieren Klassen, die Attribute (Eigenschaften) und Methoden (Funktionalität) besitzen. Über den Konstruktor kann ein neues Objekt einer Klasse erstellt werden.

38010-14
Projekt für A2 bis A6 und Aufgabe 1

A2 Programmtext analysieren

a) Öffne das hinterlegte Projekt `3-1_Simulation` und nenne alle vorhandenen Klassen, ihre Beziehungen zueinander sowie ihre Methoden in Greenfoot.
b) Nenne alle Attribute und Methoden der Klasse `Auto`. Unterscheide dabei die zugewiesenen von denjenigen, die von der Actorklasse vererbt werden.
c) Nenne die Attributwerte, die ein neues Objekt der Klasse `Auto` besitzt, wenn es mit dem Konstruktor ohne Parameter erzeugt wurde.

Anweisungen mit Java umsetzen

In Methoden können Anweisungen wie Wertzuweisungen, Methodenaufrufe oder Rechenoperationen geschrieben werden. Diese Anweisungen können darüber hinaus mit Kontrollstrukturen strukturiert werden.

A3 Methoden analysieren und vervollständigen

a) Beschreibe die Arbeitsweise der Methode `anfahren()` in eigenen Worten.
b) Passe die Methode `anfahren()` so an, dass das Auto, sobald es den rechten Rand der Welt erreicht, am linken wieder erscheint, und der Wert für den Gang um eins erhöht wird. Dieser Wert soll aber nicht höher als sechs werden können.

Neue Klassen definieren

In Greenfoot kann man nach einem Rechtsklick auf `Actor` und Klicken auf die entsprechende Schaltfläche eine neue (Unter-)klasse erzeugen. Bei der Deklaration von Attributen in Klassen muss man den zugehörigen Datentyp mit angeben.

A4 Klassen und Attribute erstellen

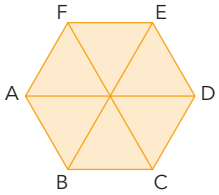
a) Erstelle eine neue Klasse `Bus` als Unterklasse von `Actor`.
b) Erweitere die Klasse `Bus` um die Attribute `hersteller`, `leistung`, `besitztAbs` und `baujahr`. Achte dabei auf sinnvolle Datentypen!

Methoden mit Ein- und Ausgaben definieren

Methoden besitzen in Java einen Namen sowie optional eine Eingabe (Parameter) und eine Ausgabe (Rückgabewert). Im Rumpf der Methode stehen Anweisungen.

A5 Methoden analysieren und programmieren

a) Füge eine Methode `fahren()` zur Klasse Bus hinzu, die den Bus ein Sechseck mit 200 Pixel Seitenlänge mit der Geschwindigkeit 10 Pixel/act fahren lässt.
b) Passe die Methode `fahren()` so an, dass der Bus sobald er an den Rand stößt seine Fahrtrichtung um einen beliebigen Winkel bis 90° nach links verändert. Zugleich soll er beim Auftreffen auf das Auto verschwinden und an beliebiger Stelle nach einer Rechtsdrehung wieder auftauchen.
c) Teste dein Programm, indem du es durch Klick auf „Run“ ausführst.



zu a) Überlege dir den Drehwinkel für ein Sechseck

Bei Klick auf „Run“ wird dein Programm ausgeführt, bei „Act“ in Einzelschritten.

Algorithmen implementieren

Löst ein beschriebener Ablauf ein allgemeines Problem, so spricht man auch von einem **Algorithmus**. Eine „allgemeine“ Lösung ist unabhängig von verschiedenen Startbedingungen oder Eingabewerten.

A6 Algorithmen entwerfen und implementieren

a) Implementiere eine Methode `tanken()`, die den Bus zu einer Tankstelle (Position x = 400 und y = 400) in der Welt fahren lässt und das Programm beendet. Bei jeder `move()`-Anweisung wird der Wert des Attributs `benzin` um fünf (Verbrauch) vermindert und, falls dieser eine bestimmte Grenze (z. B. 500 für 50 Liter) unterschreitet, die Methode `tanken()` aufgerufen.
b) Teste dein Projekt, indem du verschiedene Anfangswerte für den Benzinvorrat und den Verbrauch vorgibst. Überlege dir dazu vorab geeignete Werte und beschreibe deine Beobachtungen.

Tipp zu a): Erweitere die Klasse um ein neues Attribut. Vergiss nicht, dieses auch im Konstruktor zu initialisieren, z. B. benzin=3000 für 300 Liter

MERKE

Bei der objektorientierten Programmierung (z. B. mit Java) schreibt man Klassen. Diese definieren die Attribute und Methoden von Objekten einer bestimmten Art. Methoden bestehen aus Anweisungen und Kontrollstrukturen und können einen Algorithmus umsetzen. Neue Objekte einer Klasse werden in Java mithilfe des Konstruktors erzeugt.

AUFGABEN

1 Öffne das hinterlegte Projekt `3-1_Simulation`.
a) Ergänze die Methode `fahren()`, damit das Auto 100 Pixel bevor es den Rand erreicht seine Fahrtrichtung um 90° gegen den Uhrzeigersinn ändert.
b) Passe die Methode `fahren()` so an, dass die Richtungsänderung abwechselnd im und gegen den Uhrzeigersinn erfolgt.

RÜCKBLICK

Am Ende dieses Unterkapitels hast du wiederholt, …

- Welche Konzepte und Begriffe in der Programmierung wichtig sind.
- Klassen, Methoden und Algorithmen in Java zu implementieren.

3.2 Rückblick: Modellierung von Algorithmen, Klassen und Objekten

EINSTIEG

Lea möchte ein Spiel programmieren, in dem der Spieler durch einen virtuellen Zoo laufen kann. Dafür beginnt sie zunächst mit der Modellierung eines Raubtierhauses.

- Beschreibe, welche Klassen man zur Umsetzung des Projektes nutzen könnte.
- Nenne nötige Objekte, um einen Familienbesuch im Raubtierhaus zu modellieren.

ERARBEITUNG

Struktur mit einem Klassendiagramm modellieren

Eine Klassenkarte besteht aus dem Namen der Klasse, sowie eine Auflistung aller Attribute und Methoden dieser Klasse. Zeichnet man mehrere Klassen eines Projektes zusammen, so spricht man von einem Klassendiagramm. Mit einem Klassendiagramm kann man die Klassen in einem Programm modellieren, bevor man es implementiert. Das erlaubt zum Beispiel bei mehreren Klassen ein arbeitsteiliges Vorgehen.

Tiger
gewicht fellfarbe geschlecht
bruelle() istimkaefig() istImKaefig() wirdGefuettert()

Besucher
alter hatKamera geschlecht
fotografiere() beobachte()

A1 Modellierung eines Programms als Klassendiagramm

a) Vervollständige das oben begonnene Klassendiagramm, sodass Lea es als Grundlage ihrer Implementierung verwenden kann. Zeichne dabei die weiteren Klassen `Loewe`, `Waerter`, und `Kaefig`.

b) Notiere den Programmtext, um die Klasse `Loewe` in Java umzusetzen.

c) Nenne drei weitere Klassen, die im Programm vorkommen könnten.

d) Begründe, ob man mehrere Klassen zu einer Oberklasse zusammenfügen kann.

Situationen mit einem Objektdiagramm modellieren

Eine Objektkarte gibt den Namen und die Attributwerte des Objektes an. Im Gegensatz zur Klassenkarte besitzt diese abgerundete Ecken und nur zwei Bereiche: Die Methoden müssen nicht aufgeführt werden, da sie bei allen Objekten identisch sind. Zeichnet man mehrere Objekte zu Klassen im Projekt, so spricht man von einem Objektdiagramm. Mit ihm kann man zum Beispiel eine gewisse Programmsituation darstellen. Damit ist es häufig möglich, fehlende Informationen oder die ungeschickte Wahl von Datentypen frühzeitig zu bemerken.

max: Tiger
gewicht = 140 fellfarbe = "getigert" geschlecht = 'm'

henriette: Besucher
alter = 15 hatKamera = true geschlecht = 'w'

In diesem Buch werden zur besseren Unterscheidung Klassenkarten blau und Objektkarten rot dargestellt.

A2 Modellierung einer Situation als Objektdiagramm

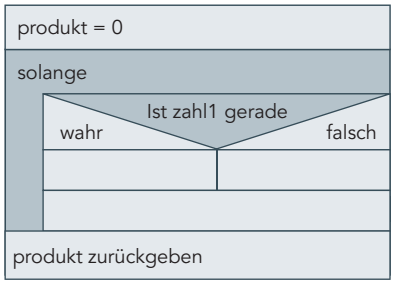
a) Vervollständige das oben begonnene Objektdiagramm, sodass die folgende Situation beschrieben wird: Lea und ihre Zwillingsschwester Henriette beobachten die Fütterung der Tiger `Max` und `Moritz` durch den Wärter `Leo`

b) Gib den Programmtext an, mit dem das `Max`-Objekt erzeugt werden kann.

c) Beschreibe eine Situation, die mit den vorhandenen Klassen nicht umgesetzt werden kann.

Algorithmen mit einem Struktogramm modellieren

Da Algorithmen zur Umsetzung häufig viele Zeilen Programmtext benötigen, modelliert man sie zunächst in einem Struktogramm. In diesem erkennt man schnell die wesentlichen Kontrollstrukturen. Darüber hinaus ist es möglich, einzelne Abschnitte in einer kurzen Beschreibung, statt einem Programmtext umzusetzen. Auf diese Weise kann man schrittweise von einer Grobstruktur zu einer genauen Umsetzung kommen.



A3 Algorithmen entwerfen und implementieren

a) Vervollständige das oben begonnene Struktogramm zu einem Algorithmus, der zwei Zahlen nach der „Ägyptischen Methode" multipliziert. Ist `zahl1` ungerade, addiert man `zahl2` zum Produkt. Solange `zahl1` größer ist als 1, halbiert man sie und rundet das Ergebnis ab, `zahl2` wird verdoppelt.

b) Gib den Java-Programmtext für eine Implementierung des Algorithmus an, und führe ihn auf dem Papier für zwei selbstgewählte Zahlen durch.

MERKE

In der Informatik wird ein Programm vor seiner Umsetzung zunächst modelliert. Für die Modellierung der Struktur verwendet man das Klassendiagramm, für die Modellierung einer speziellen Situation im Programm das Objektdiagramm.
Der Ablauf eines Algorithmus, der umgesetzt werden soll, kann durch ein Struktogramm veranschaulicht werden.

AUFGABEN

1 In der Nähe des Zoos gastiert ein kleiner Zirkus, der auch Raubtieraufführungen und weitere Attraktionen anbietet.

a) Nenne zwei weitere Gründe, warum es sinnvoll ist, ein Programm vor der Implementierung zunächst als Klassendiagramm zu modellieren.

b) Nenne zwei weitere Gründe, eine bestimmte Programmsituation vor der Implementierung zunächst als Objektdiagramm zu modellieren.

c) Sammle diejenigen Klassen, die für die Darstellung einer Vorstellung nötig wären.

d) Zeichne diese Klassen im Klassendiagramm.

e) Erstelle aus diesem Klassendiagramm ein mögliches Objektdiagramm für eine Raubtiernummer mit je zwei Löwen und Tigern.

2 Nimm fundiert Stellung zu der Aussage: „Eine Klasse ist die Menge aller Objekte mit gleichen Attributen und Methoden."

3 Zeichne ein Struktogramm zu folgenden Algorithmus:
Um eine Dezimalzahl in eine Dualzahl umzuwandeln, dividiert man diese wiederholt durch 2 mit Rest bis zum Ergebnis 0. Die entstehenden Reste (0 oder 1) ergeben rückwärts gelesen die Dualzahl.

Hinweis:
Die Division zweier ganzer Zahlen (int-Variable) liefert stets den ganzzahligen Anteil. Mit dem Modulo-Operator % bestimmt man den Rest.
z. B. liefert 14%3 den Wert 2

RÜCKBLICK

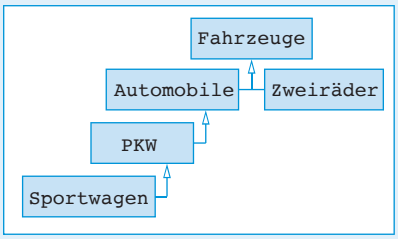
Am Ende dieses Unterkapitels hast du wiederholt, ...

- Klassen- und Objektdiagramme sowie Struktogramme von Algorithmen zu zeichnen.
- Programmzustände mit Klassen- und Objektdiagrammen zu modellieren.

3.3 Rückblick: Vererben und Überschreiben

EINSTIEG

Jana hat die nebenstehende Darstellung der Fahrzeughierarchie entdeckt und möchte diese nun in Java übertragen.



- Beschreibt am Beispiel Automobile/ PKW und Zweiräder gemeinsame Attribute und begründet, warum Vererbung hier sinnvoll ist.
- Nenne fünf weitere sinnvolle Unterklassen und begründe jeweils, von welcher vorhandenen Klasse diese erben sollten.

ERARBEITUNG

Ober- und Unterklassen

Bei der Vererbung gibt es Unter- und Oberklassen. Eine Unterklasse besitzt automatisch alle Attribute und Methoden der Oberklasse. Es werden zwei Richtungen unterschieden: Wird eine Unterklasse erstellt, so spricht man von einer Spezialisierung der Oberklasse. Umgekehrt ist die Oberklasse eine Generalisierung der Unterklasse.

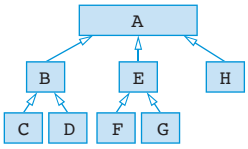
A1 Vererbung analysieren

a) Gib Beispiele für Attribute und Methoden an, die von der Klasse `PKW` auf die Unterklassen vererbt werden.

b) Finde Attribute und Methoden, die nur in einer der Unter-, aber nicht in der zugehörigen Oberklasse vorkommen.

c) Beschreibe in eigenen Worten allgemein den Zusammenhang zwischen Ober- und Unterklassen, ohne das obige Beispiel zu verwenden.

Vererbungshierarchien



Ist eine Unterklasse gleichzeitig auch Oberklasse einer weiteren Klasse, spricht man von mehrstufiger Vererbung und es entsteht eine Vererbungshierarchie.

A2 Vererbungshierarchien analysieren und modellieren

a) Nenne alle Klassen im oben gezeichneten Klassendiagramm, die sowohl Unter-, als auch Oberklasse einer anderen Klasse sind.

b) Beschreibe eine Situation (nicht Fahrzeuge), die mit dieser Vererbungshierarchie modelliert werden kann.

c) Zeichne ein Klassendiagramm mit folgenden Unterklassen der Oberklasse `Tier`: `Wirbeltiere, WirbelloseTiere, Fische, Amphibien, Säugetiere, Vögel, Reptilien, Insekten und Spinnen`.
Beachte dabei folgende Zusammenhänge: Säugetiere sind Wirbeltiere mit Haaren, Vögel sind Wirbeltiere mit Federn, Amphibien sind wechselwarme Wirbeltiere, die an Land und im Wasser leben, Fische sind Wirbeltiere, die Eier legen und im Wasser leben, Insekten sind wirbellose Tiere mit Panzer und Spinnen sind wirbellose Tiere mit acht Beinen.

d) Nenne eine weitere Klasse und ihre Oberklasse, die dem Klassendiagramm hinzugefügt werden könnte, sodass die Vererbungshierarchie vierstufig wird.

Vererbung und Überschreiben

Definiert man eine Methode in der Unterklasse erneut, so wird die ursprüngliche Methode überschrieben. Sie besitzt hier eine andere Funktionsweise als auf der Oberklasse.

Eine Methode, die in Greenfoot besonders häufig überschrieben wird, ist die `act` Methode von `Actor`. Diese wird regelmäßig aufgerufen, wenn auf die Schaltfläche „Ausführen“ geklickt wird.

A3 Methoden analysieren und programmieren

a) Öffne das Projekt `3-3_Blumen`. Erstelle eine neue Klasse `Tulpe` als Unterklasse von `Actor`.
b) Beschreibe das Resultat des Aufrufens einer überschriebenen Methode.
c) Überschreibe die Methode `act()` der Klasse `Tulpe`, sodass sie zunächst wächst, dann blüht und dann verwelkt. Verwende dazu verschiedene Bilder.

38010-16
Projekt für A3, A4

Die Annotation Override

Eine Methode überschreibt automatisch eine Methode der Oberklasse, wenn diese den gleichen Namen sowie die gleiche Ein- und Ausgabe besitzt. Ändert man jedoch den Methodennamen in der Oberklasse, so wird diese Verbindung aufgelöst. Die Methode der Unterklasse überschreibt nun keine Methode der Oberklasse mehr. Das kann schnell versehentlich passieren, wenn man nicht auf Unterklassen achtet. Deshalb kann man in der Unterklasse eine Anmerkung (Annotation) in den Programmtext einfügen. Die Annotation `@Override` kann vor eine Methode geschrieben werden. Diese Annotation sagt dem Übersetzer, dass diese Methode eine gleichnamige Methode der Oberklasse überschreibt. Ist das nicht der Fall, etwa, weil es keine gleichnamige Methode in der Oberklasse gibt, bricht der Übersetzer mit einem Fehler ab.

```
@Override
public void act()
{
}
```

A4 Private Methode definieren und verwenden

a) Benenne in der Klasse Blumen die Methode `bluehen()` in `bluehen2()` um. Führe das Projekt aus und beschreibe, was du beobachtest.
b) Füge die Annotation `@Override` vor der Methode `bluehen()` in der Unterklasse `Tulpe` ein. Beschreibe den auftretenden Fehler.

MERKE

In Greenfoot werden alle Klassen als Unterklassen der World- bzw. Actorklasse angelegt und erben somit alle Methoden und Attribute von ihnen. In der Hierarchie erbt jede Unterklasse alle Methoden ihrer Oberklasse und vererbt sie weiter an ihre Unterklassen. Vererbte Methoden können in Unterklassen neu angelegt werden, die Originalmethode also überschrieben werden. Die gleichnamigen Methoden besitzen dann in der jeweiligen Klasse unterschiedliche Funktionen.

AUFGABEN

1 Für einen Stammbaum gibt es die Klassenhierarchie Opa → Vater → Sohn.
a) Gib Gründe an, warum diese Hierarchie wenig sinnvoll ist.
b) Konstruiere eine passende Hierarchie für eine Familie mit drei Generationen.

2 Lege ein neues Projekt `Tierheim` mit den Tierklassen `Hund` und `Katze` als Unterklassen von `Actor` mit je drei sinnvollen Attributen an.
a) Im Tierheim fallen tägliche Arbeiten wie Füttern an, die alle Tiere betreffen, aber auch spezielle für Hunde wie „Gassigehen". Implementiere drei Methoden, die alle Tiere betreffen, und an die Unterklassen vererbt werden.
b) Überschreibe je eine Methode für die Unterklassen `Hund` und `Katze`, um sie besser an die jeweilige Tiergattung anzupassen.

RÜCKBLICK

Am Ende dieses Unterkapitels hast du wiederholt, ...

- Vererbung und den Zusammenhang zwischen Ober- und Unterklassen,
- die Möglichkeit des Überschreibens von Methoden in Unterklassen.

EINSTIEG

Jana sagt: „Wenn ich eine Variable deklariere, so muss ich dieser einen Namen und Datentyp zuweisen."
Lukas korrigiert: „Ja, aber zusätzlich braucht eine Variable auch einen Wert!"

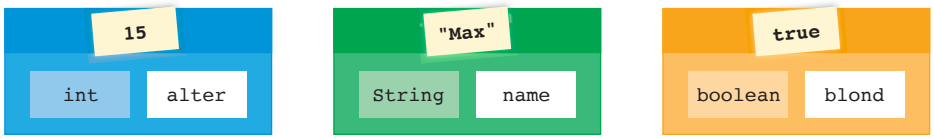
- Nennt die Anweisung, mit der man in Java ein Objekt erstellen kann.
- Diskutiert, ob ein Objekt in einer Wertzuweisung eher als Namen, Datentyp, oder Wert benutzt wird.

ERARBEITUNG

Visualisieren von Variablen

Eine Variable hat bekanntermaßen einen Datentyp und einen Wert. Diese kann man auch grafisch darstellen. Dafür beschriftet man einen Behälter mit Namen und Datentyp der Variablen und legt einen Zettel mit dem Wert hinein:

Häufig werden unterschiedliche Typen in unterschiedlichen Farben dargestellt.




```
int alter = 15;
```

```
String name = „Max"; boolean blond = true;
```

A1 Variablen visualisieren

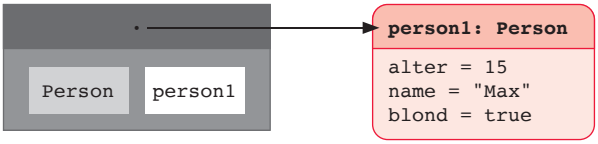
Visualisiere eine Variable mit Namen `pi` und Wert `3.1415` im Behältermodell.

Klassen als Datentypen

Erzeugt man ein neues Objekt, so kann dieses ebenfalls in einer Variablen gespeichert werden. Dabei ist die Klasse der Datentyp der Variable. Das Objekt steht an der Stelle des Wertes. In der Visualisierung verknüpft man die Objektkarte mit dem Zettel:

Diese Verknüpfung wird technisch als Referenz (➥ 4.1) umgesetzt.

In Java werden Objekte über den Konstruktor erstellt. Dieser wird entweder über die Greenfoot-Oberfläche oder mit der Anweisung `new` aufgerufen.




```
Person person1 = new Person(15, "Max", true)
```

A2 Objekte als Variablen visualisieren

a) Zeichne eine Klassenkarte zur Klasse `Person`.
b) Begründe, an welcher Stelle in den Zeichnungen du etwas ändern müsstest, wenn die Programmzeile stattdessen folgendermaßen lauten würde:
`Person person2 = new Person("Moritz", 15, true)`
c) Zeichne ein weiteres Objekt der Klasse `Person` in obiger Visualisierung.

Typprüfung in Java

Die genaue Fehlermeldung lautet:
„incompatible types: double cannot be converted to String"
Warum sie so formuliert ist wird im nächsten Kapitel erklärt.

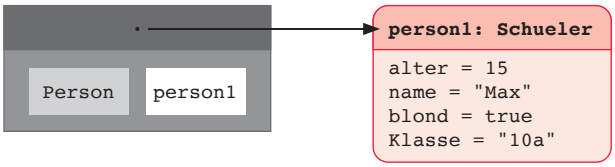
In Java können nur Werte in Variablen gespeichert werden, die zu diesem Datentyp passen. Dies wird von der **Typprüfung** sichergestellt, die bei jedem Übersetzungsvorgang durchgeführt wird. So ist es etwa nicht möglich, in die Variable `int alter` den Wert `"Max"` zu speichern. Stattdessen würde die Anweisung zu einem Fehler führen.

A3 Typprüfung nachvollziehen

Begründe Probleme beim Speichern von unpassenden Werten in einer Variable.

Objekte der Unterklasse als Oberklasse speichern

Die Typprüfung verhindert, dass unpassende Werte in einer Variablen gespeichert werden. Doch nicht immer ist ein anderer Wert auch unpassend. So ist es zum Beispiel möglich, Objekte der Unterklasse in Variablen der Oberklasse zu speichern:




```
Person person1 = new Schueler(15, "Max", true, "10a");
```

A4 Objekte von Unterklassen als Variablen visualisieren

a) Zeichne ein Klassendiagramm mit den Klassen `Person` und `Schueler` zu obiger Visualisierung.

b) Begründe, warum man in obiger Visualisierung im Behälter links den Datentyp von Person nicht weglassen kann.

MERKE

Variablen können durch einen mit Datentyp und Name beschrifteten Behälter visualisiert werden. Der Wert der Variable wird durch einen Zettel im Behälter angegeben. Auch Objekte kann man auf diese Weise visualisieren: Die Objektkarte wird mit dem Zettel verknüpft und die Klasse ist der Datentyp des Objektes. Die **Typprüfung** verhindert die Speicherung von unpassenden Werten in einer Variablen. Es ist trotzdem möglich, Objekte von Unterklassen in Behältern der Oberklasse zu speichern.

AUFGABEN

1 Gegeben sei folgender Programmtext:

```
Artikel artikel;
artikel = new Buch("Robinson Crusoe");
artikel = new Film("Robinson Crusoe");
```

a) Zeichne ein zum Programmtext passendes Klassendiagramm.

b) Zeichne die Variable `artikel` nach der zweiten und dritten Anweisung.

c) Begründe, welchen Vorteil es haben kann, Objekte der Unterklasse in einer Variablen der Oberklasse zu speichern.

2 Visualisiere folgende Variablen, wenn der gespeicherte Wert für diese Variable erlaubt ist. Zeichne falls nötig auch das zugehörige Klassendiagramm.

a) `boolean istNull = 0;`

b) `String name = "1234";`

c) `Person person = "Moritz";`

d) `Tier amsel = new Drossel();`

RÜCKBLICK

Am Ende dieses Unterkapitels hast du gelernt, ...

- Klassen als Datentypen zu interpretieren.
- Variablen mit einem Behälter zu visualisieren.

EINSTIEG

Theodora möchte ein Computerprogramm zum Verschlüsseln von Nachrichten schreiben. Bei einer Recherche ist sie auf die Cäsar-Verschlüsselung gestoßen. Dabei wird jeder Buchstabe eines Textes um eine bestimmte Anzahl an Stellen im Alphabet verschoben. So wird das Wort „`INFORMATIK`“ mit dem Schlüssel 13 zu „`VASBEZNGVX`“ kodiert.

- Kodiere das Wort „Biber“ mit dem Schlüssel 3.
- Beschreibe, was beim Verschlüsseln von „Ziel“ beachtet werden muss.

ERARBEITUNG

Typumwandlung

Bei Theodoras Beispiel lässt sich die Verschiebung eines Buchstaben als Addition darstellen. Das ist auch in Java möglich: `int i = 'A' + 8;`
Auf beiden Seiten des Operationszeichens stehen unterschiedliche Datentypen. Damit die Addition dennoch funktioniert, muss der Buchstabe zunächst in eine Zahl umgewandelt werden. Das ist mit einer **Typumwandlung** möglich. Dabei wird der zu verändernde Wert in einen analogen Wert mit anderem Datentyp umgewandelt:

Der umgewandelte Wert ist dabei streng genommen eine eigene Variable, besitzt aber keinen eigenen Namen.

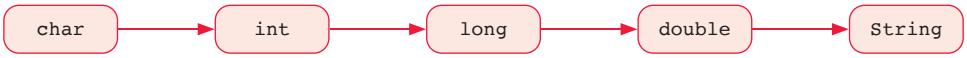
Typumwandlung	Startwert	Endwert
int → double	3	3.0
int → String	42	"42"
int → long	300	300
char → int	'A'	65

A1 Typumwandlung erkennen und beschreiben

a) Begründe, warum die Anweisung 'A' + 8 eine Zahl als Ergebnis liefert.
b) Beschreibe, was passieren muss, damit man das Ergebnis dieser Anweisung als Buchstaben speichern kann.
c) Nenne weitere Beispiele, bei denen eine Typumwandlung auftritt.

Implizite Typumwandlung

Beim obigen Beispiel handelt es sich um eine **implizite Typumwandlung**, weil diese im Programmtext nicht ausdrücklich angegeben ist, sondern automatisch durchgeführt wird. Das ist bei bestimmten Umwandlungen möglich, wenn dabei keine Informationen verloren gehen. Genauer werden in Java die folgenden Datentypen bei Bedarf automatisch ineinander umgewandelt:



Jeder Datentyp kann direkt in den weiter rechts stehenden Datentyp umgewandelt werden. Ein `int` wird also z. B. ohne Umweg in einen `String` umgewandelt.

38010-17
Projekt für A2, A3 und Aufgabe 1

A2 Implizite Typumwandlung anwenden

Öffne das Projekt `3-5_Caesar` und darin die Klasse `Verschluesselung`.

a) Nenne jede Typumwandlung in der Methode `codiereBuchstabe()`.
b) Recherchiere im Internet, warum der Buchstabe `'A'` der Zahl `65` entspricht. Benutze für die Recherche auch den Suchbegriff ASCII.
c) Erweitere die Methode, sodass auch Kleinbuchstaben richtig kodiert werden.

Explizite Typumwandlung

Neben der impliziten gibt es in Java auch eine **explizite Typumwandlung**. Diese wird explizit als Anweisung im Programmtext angegeben. Hierfür muss man den gewünschten Zieltyp in runden Klammern vor die umzuwandelnde Variable schreiben:

```
char c = (char) 65; // 'A'
```

Einige Typumwandlungen sind in Java nur explizit möglich, da bei der Umwandlung Informationen verloren gehen können. Beispiele sind in der Tabelle orange hinterlegt:

Typumwandlung	Startwert	Endwert
double → int	3.0	3
	3.5	3
int → char	65	'A'
	9651	'△'
long → int	300	300
	4294967298	2

In einzelnen Fällen ist eine explizite Typumwandlung gar nicht möglich. So können `boolean` und `String` nicht in einer expliziten Typumwandlung benutzt werden.

Zur Übersicht ist es häufig sinnvoll, jede Typumwandlung explizit zu machen, auch wenn diese implizit benutzt werden kann.

A3 Explizite Typumwandlung anwenden

a) Vervollständige die Methode `codiereWort()`. Setze dabei implizite und explizite Typumwandlungen bewusst ein.

b) Beurteile folgende Aussage: „Ein Computer kann nur mit Zahlen rechnen!“

Hinweis zu a):
Du kannst zwar einen String nicht direkt in eine Zahl umwandeln. Es ist aber möglich, einen Buchstaben an einer bestimmten Stelle im String zu bekommen!

MERKE

Durch **implizite** und **explizite Typumwandlung** lässt sich der Datentyp einer Variablen so anpassen, dass Operationen und Methoden mit den notwendigen Datentypen durchgeführt werden können.

AUFGABEN

1 Die Cäsar-Verschlüsselung ist nicht sehr sicher, da jeder Buchstabe immer gleich verschlüsselt wird. Eine einfache Möglichkeit, den Schlüssel zu raten, ist die Zählung der Buchstaben und die Annahme, dass der am häufigsten vorkommende Buchstabe ursprünglich ein E war. Das soll nun automatisch geschehen.
Achte bei deiner Implementierung immer auf korrekte Typumwandlungen und teste deine Methoden.

a) Vervollständige die Methode `zaehleBuchstabeInWort()`.

b) Nenne alle Typumwandlungen, die im vorgegebenen Programmtext der Methode `gibHaeufigsterBuchstabe()` stehen. Begründe, ob es sich dabei um implizite oder explizite Typumwandlungen handelt.

c) Vervollständige die Methode `gibHaeufigsterBuchstabe()`: Die Methode soll für jeden Buchstaben prüfen, ob er häufiger vorkommt als der bisher häufigste Buchstabe. Wenn ja, soll der häufigste Buchstabe auf den aktuellen Buchstaben gesetzt und die Variable `anzahl` aktualisiert werden.

d) Definiere eine Methode `rateSchluessel()`. Diese soll ein Wort entgegennehmen und den vermuteten Schlüssel als Zahl zurückgeben. Die Methode rät dabei, dass der häufigste Buchstabe ursprünglich ein E war.

Hinweis zu 1:
Da die Methode codiereWort nicht mit Leerzeichen umgehen kann, musst du die Wörter zum Testen im CamelCase schreiben!

RÜCKBLICK

Am Ende dieses Unterkapitels hast du gelernt, ...

- Typumwandlungen mit Variablen durchzuführen.
- zwischen expliziter und impliziter Typumwandlung zu unterscheiden.

EINSTIEG

Yusuf hat die Methode `wasPassiertHier()` entworfen. Er ruft die Methode auf und erhält die unter der Methode stehende Bildschirmausgabe.

```
void wasPassiertHier()      {
      System.out.println(3 + 5);
      System.out.println("3" + "5")
}
```

```
8
35
```

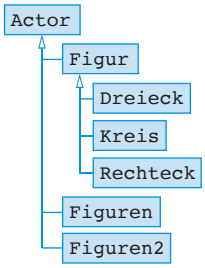
- Sandra vermutet, dass hier eine Typumwandlung vorliegt. Begründe, warum sie nicht recht hat!
- Überlegt, in welcher Art und Weise der Plus-Operator dennoch ein unerwartetes Verhalten zeigt.

ERARBEITUNG

38010-18
Projekt für A1 bis A4

A1 Objekte deklarieren und initialisieren

Öffne das hinterlegte Projekt `3-6_Figuren` und beschreibe die Beziehungen zwischen den einzelnen Klassen.



a) Erzeugt man ein Objekt der Klasse `Figuren`, werden Objekte verschiedener Klassen angelegt. Prüfe anhand des Programmtextes, welches es genau sind.

b) Begründe, warum in die Variable `f2` ein Objekt des Typs Dreieck gespeichert werden darf.

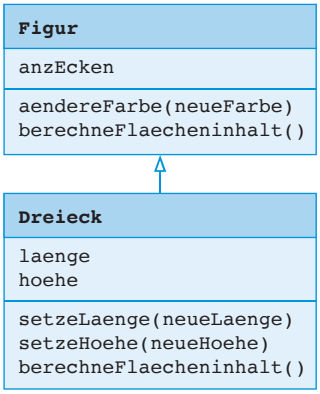
Polymorphismus bei höheren Datentypen

Speichert man ein Objekt der Unterklasse in einer Variablen vom Datentyp der Oberklasse, so ist es nicht mehr möglich, Attribute und Methoden zu nutzen, die nur auf der Unterklasse definiert sind. Stattdessen wird das Objekt der Unterklasse wie sein Behälter (mit dem Datentyp der Oberklasse) behandelt. Hier hat also eine implizite Typumwandlung stattgefunden. Diese spezielle Art der Typumwandlung, bei der ein Objekt gleichzeitig mehrere Typen hat, bezeichnet man auch als **Polymorphie**.

Polymorphie kommt aus dem Griechischen und bedeutet übersetzt „Vielartigkeit".

A2 weitere Objekte polymorph anbieten

Das rechts dargestellte Klassendiagramm ist ein Ausschnitt aus dem Projekt `3-6_Figuren`.



a) Beschreibe mithilfe des Klassendiagramms, auf welche Attribute das Objekt `d1` bzw. `f2` aus dem Programmtext von Aufgabe 1 Zugriff hat.

b) Erkläre, aus welcher Klasse die Methode für die Ausführung der folgenden Programmzeile genutzt wird. `f2.berechneFlaecheninhalt();`

c) Begründe, warum die folgende Programmzeile zu einer Fehlermeldung führt.

```
Dreieck d2 = new Figur();
```

A3 den Datentyp eines Objekts festlegen

Der rechts abgebildete Programmtext ist Teil der Klasse `Figuren2` im Projekt `3-6_Figuren`.

```
Figur f;
if(zufall() < 0.5) {
    f = new Dreieck();
}  else{
    f = new Rechteck();
}
fügeInWeltEin(f);
```

a) Beschreibe mit eigenen Worten, was für die Initialisierung von Figur f zu beachten ist.

b) Erkläre den Unterschied im Vergleich zu A1.

Hinweis: Die Methode zufall() gibt eine Zahl zwischen 0.0 und 1.0 zurück.

Dynamische Bindung von Objekten

Wird ein Objekt der Unterklasse im Behälter der Oberklasse gespeichert, so sind nur die Methoden der Oberklasse verfügbar. Diese können jedoch von den Unterklassen überschrieben werden. Aus diesem Grund muss während der Ausführung des Programms geprüft werden, ob die Methode überschrieben ist. Ist das der Fall, so wird die auszuführende Methode der Unterklasse ermittelt und ausgeführt. Diese Art der Vermittlung nennt man auch **dynamische Bindung**.

A4 Dynamische Bindung erkunden

Die dargestellte Methode befindet sich in der Klasse `Figuren2`.

```
viod tanzen() }
    f3 = r2;
    f3.tanzen();
    f3 = d2;
    f3.tanzen();
    f3 = r2;
    f3.tanzen();
}
```

a) Erkläre, wo hier dynamische Bindung auftritt.
b) Beschreibe, was beim Aufruf dieser Methode passiert.
c) Schreibe eine eigene Methode in der Klasse `Figuren2`, in der mehrere Objekte dynamisch gebunden werden.
d) Begründe, warum eine dynamische Bindung nur bei einer Methode möglich ist, die in Unterklassen überschrieben wurde.
e) Begründe, warum nicht jedes Überschreiben von Methoden zu einer dynamischen Bindung führt.

MERKE

Bei einer Variablen, die eine Klasse als Datentyp hat, können Objekte **polymorph** angeboten werden, indem das Objekt der Unterklasse in einer Variable vom Datentyp der Oberklasse abgespeichert wird. In diesem Fall kann man nur Attribute und Methoden der Oberklasse aufrufen, auch wenn die der Unterklasse nach wie vor die eigenen besitzt. Wird eine Methode überschrieben, so muss die entsprechende Methode der Unterklasse über **dynamische Bindung** ermittelt werden.

AUFGABEN

1 Erstelle unter Verwendung der Vererbungsbeziehung ein Projekt mit den Klassen `Tier`, `Känguru`, `Frosch` und `Delfin`. Nutze für die Unterklassen ein passendes Bild.
a) Veranschauliche im Klassendiagramm die Beziehung zwischen den Klassen.
b) Ergänze in allen Klassen eine Methode `huepfen()`, welche die Figuren unterschiedlich bewegen lässt.
c) Erzeuge eine Klasse Wiese mit mehreren Tieren. Nutze die dynamische Bindung von Objekten, um die Tiere nacheinander hüpfen zu lassen. Nutze dafür die Act-Methode der Klassen.

2 Gegeben sind Flächeninhalt und Grundseitenlänge eines Dreiecks. Gesucht ist ein flächengleiches Rechteck, dessen Breite so lang ist wie die Dreiecksgrundseite.
a) Informiere dich in den einzelnen Klassen des `Figuren`-Projekts, welche Methoden für das Zeichnen des flächengleichen Rechtecks hilfreich sind.
b) Vervollständige die Programmzeilen der Methode `RechteckZeichnen()`.

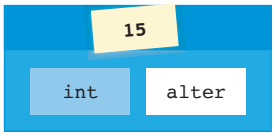
RÜCKBLICK

Am Ende dieses Unterkapitels hast du gelernt, ...

- Objekte polymorph anbieten zu können.
- den Datentyp eines Objekts zur Laufzeit dynamisch festzulegen.

Visualisierung von Variablen ↪ 3.4

Variablen besitzen einen Namen und einen Wert. Visualisiert werden Variablen häufig durch einen farbigen Behälter, der einen Zettel mit dem Wert enthält.




```
int alter = 15;
```

Klassen als Datentypen ↪ 3.4

Der Datentyp von Objekten ist die Klasse, aus der das Objekt erzeugt wird. Das durch die Anweisung

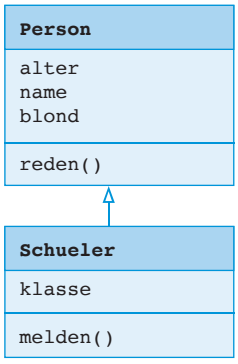
```
Person person1;
```

deklarierte Objekt besitzt also die Klasse `Person` als Datentyp und `person1` als Objektnamen.

Objekte einer Unterklasse können dabei auch in einer Variablen der Oberklasse gespeichert werden. Die Anweisung

```
person1 = new Schueler(…);
```

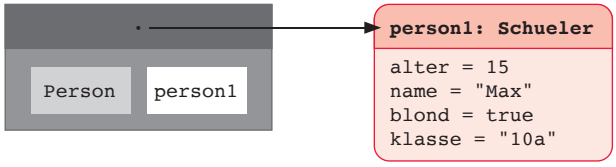
erzeugt also ein neues Objekt der Klasse Schueler, das in einer Variable der Klasse `Person` gespeichert wird.



Diese Vererbungsstruktur muss vorliegen, damit die Anweisungen auf der linken Seite gültig sind.

Objekte mit Datentyp visualisieren ↪ 3.4

Objekte können wie Variablen visualisiert werden. Dabei wird ein Kasten für die Variable erstellt, die durch einen Pfeil mit der Objektkarte verknüpft wird.




```
Person person1 = new Schueler(15, "Max", true, "10a");
```

Typumwandlung

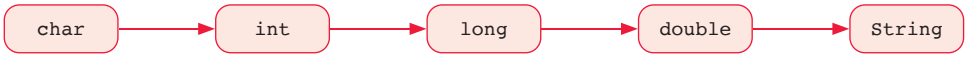
Manche Datentypen können in andere Datentypen umgewandelt werden. So werden etwa Buchstaben im Computer eigentlich als Zahlen gespeichert.

char	.	0	1	=	A	B	a
int	46	48	49	61	65	66	97

Ausschnitt aus der ASCII-Tabelle.
Diese beschreibt die häufigste Möglichkeit, wie man Buchstaben als Zahlen speichern kann.

Implizite Typumwandlung

Manche Typen können ohne Anweisung automatisch ineinander umgewandelt werden. In diesem Fall spricht man von einer impliziten Typumwandlung.



So wird etwa in der Anweisung `'A' + 3` das `'A'` automatisch in einen `int` umgewandelt. Das Ergebnis ist die Zahl 68. In der Anweisung `'A' + "Text"` wird `'A'` automatisch in einen `String` umgewandelt. Das Ergebnis ist der Text `"AText"`.

Explizite Typumwandlung

Eine Typumwandlung kann auch durch eine spezielle Anweisung durchgeführt werden. In diesem Fall spricht man von einer expliziten Typumwandlung.
Eine explizite Typumwandlung ist notwendig, wenn bei der Umwandlung Informationen verloren gehen können.

```
int umwandlung1 = (int) 3.14; // 3
char umwandlung2 = (char) 61; // '='
int umwandlung3 = (int) 'A'; // 65
```

Bei der dritten Umwandlung gehen keine Informationen verloren. Daher wäre diese Umwandlung auch implizit möglich.

Polymorphismus

Objekte der Unterklasse können auch in einer Variable vom Datentyp der Oberklasse gespeichert werden. Das Objekt der Unterklasse hat somit mehrere Typen und man spricht von Polymorphie. Bei polymorph gespeicherten Objekten können nur die Methoden aufgerufen werden, die auf dem Klassentyp definiert sind.
Im Beispiel links ist also etwa die Anweisung `person1.reden()` erlaubt, da `reden()` in `Person` definiert ist. Für die Anweisung `person1.melden()` müsste man hingegen `person1` als `Schueler` speichern.

Dynamische Bindung

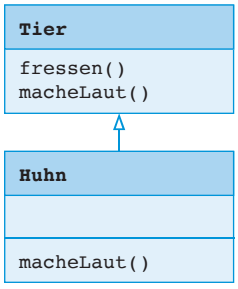
Bei der Anweisung

```
Tier huhn = new Huhn();
```

wird ein Objekt der Unterklasse in einer Variable vom Datentyp der Oberklasse gespeichert. Mit

```
huhn.macheLaut();
```

wird anschließend eine überschriebene Methode aufgerufen.
Bei der Programmausführung wird dennoch automatisch die richtige Methode der Unterklasse `Huhn` ermittelt und ausgeführt. Das bezeichnet man als dynamische Bindung.



Dynamische Bindung tritt in Java auf, wenn eine überschriebene Methode auf einem Objekt der Oberklasse aufgerufen wird.

↪ 3.1

38010-20
Projekt für Aufgabe 1

1 Öffne das `Projekt 3-8_Aquarium` und bearbeite die Arbeitsaufträge.

a) Nenne Attribute (inkl. Typen), welche die Klasse `Fisch` zur Verfügung stellt.
b) Beschreibe den Konstruktor der Klasse `Fisch` und dessen Parameter.
c) Welche Methoden stellt die Klasse `Fisch` zur Verfügung?
d) Erweitere die Klasse `Fisch` nach folgenden Vorgaben:
   - Schreibe eine Methode `void setzeTarnung(boolean tarnung)`, die den Wert des Attributs `tarnung` gemäß des Übergabeparameters verändert.
   - Schreibe eine Methode `boolean nenneTarnung()`, die den Wert des Attributs `tarnung` zurückgibt.
   - Schreibe eine Methode `void tarnen()`, die den Wert des Attributs `tarnung` zu `true` ändert.
e) Erweitere das Programm um eine Klasse `Hai`, die von `Actor` erben soll:
   - Jeder Hai hat eine Art, ein Alter und ist hungrig oder satt. Überlege dir jeweils einen passenden Datentyp und deklariere die Attribute.
   - Ein neu erzeugter Hai ist standardmäßig hungrig und ein Jahr alt. Außerdem soll die Art gemäß eines Übergabeparameters gesetzt werden.
   - Es gibt eine Methode `void bewegen(int x, int y)`, die den Hai um die gegebenen Schrittweiten in x- und y-Richtung bewegt.
   - Erweitere die `act`-Methode von `Fisch`, sodass dieser verschwindet, wenn er nicht getarnt ist und auf einen Hai trifft.

Achte darauf, welche Methoden die Klasse Fisch schon besitzt!

↪ 3.1

38010-21
Projekt für Aufgabe 2

2 Bauer Hintermoser putzt seine Schweine immer mit Gras ab, bevor er sie in den Stall bringt. Dabei braucht er für jedes Schwein einen Grasbüschel. Betrachte zunächst die Attribute und Methoden der Klassen Bauer und Schwein.

a) Öffne das hinterlegte Projekt `3-8_Schweineputzen`.
b) Ändere die `act()` Methode in der Klasse Bauer so ab, dass beim Drücken der Taste g die Methode `grasAufsammeln()` und beim Drücken der Taste p die Methode `schweinPutzen()` aufgerufen wird.
c) Schweine rennen gerne herum und wälzen sich im Matsch. Implementiere dieses Verhalten in der `act()` Methode der Klasse `Schwein`.
d) Erweitere das Programm, sodass eine Variable `grasGesammelt` um 1 erhöht wird, wenn der Bauer Gras aufsammelt und um 1 reduziert wird, wenn er ein Schwein putzt. Aktualisiere auch die Darstellung des Gras-Zählers.
e) Ändere die Methode `schweinPutzen()`, sodass der Bauer nur dann putzen kann, wenn das Schwein dreckig ist und er genügend Gras besitzt.
f) Der Bauer trägt einen Beutel bei sich, der höchstens 5 Grasbüschel tragen kann. Wenn der Beutel bereits voll ist, kann der Bauer keine weiteren Grasbüschel mehr aufnehmen. Implementiere dieses Verhalten durch Anpassen des Quelltextes der Methode `grasAufsammeln()`.
g) Sind nicht mehr genügend Grasbüschel mehr in der Welt vorhanden, kann der Bauer durch Betätigung des Brunnens den Boden wässern. Dadurch wird die Anzahl der vorhandenen Grasbüschel in der Welt wieder auf 10 aufgefüllt. Ändere die `act()` Methode in der Klasse `Bauer` so ab, dass beim Drücken der Taste b die Methode `brunnenBetaetigen()` aufgerufen wird.
h) Bei Betätigung des Brunnens wird in der Welt die Methode `bodenWaessern()` aufgerufen. Implementiere die Methode so, dass die Anzahl der Grasbüschel in der Welt wieder 10 beträgt.

Hinweis:
Benutze für e) ein neues Attribut in der Klasse Schwein.

3 Betrachte folgenden Programmtext der Klasse `Katze`: ↪ 3.1

```
class Katze
{
        String name;
        String alter;
        boolean geschlecht;

        public Katze(String nameN, int alterN)
        {
              name = "Minka";
              alter = alterN;
              geschlecht = "weiblich";
        }
 }
```

Im Programmtext sind einige syntaktische und semantische Fehler. Gib die Fehler an und begründe, um welche Art von Fehler es sich handelt. Beschreibe, wie man den jeweiligen Fehler beheben kann.

4 Begründe, ob es sich bei den folgenden Beschreibungen um einen Algorithmus handelt oder nicht, und ob eine allgemeine Lösung existiert. ↪ 3.1

a) Ein beliebiger Geldbetrag soll mit möglichst wenig Scheinen und Münzen bezahlt werden.

b) Die Lösung des Gleichungssystems $2x - 3y = 5$ und $2x - ay = 7$ wird mit dem Additionsverfahren bestimmt.

5 Öffne das hinterlegte Projekt `3-8_Hotel`. ↪ 3.2

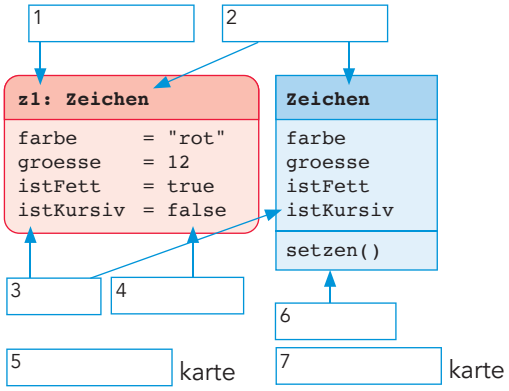
a) Erstelle ein Klassendiagramm zum Programmtext. Nenne in den Klassenkarten alle Attribute und Methoden.

b) Beschreibe folgende Situation mit einem Objektdiagramm: Der Chef André beaufsichtigt das Zimmermädchen Verena, das gerade den schmutzigen Raum 211 putzt, weil der Gast Manuel unzufrieden ist.

c) Begründe, welche Aspekte der Beschreibung nicht im Objektdiagramm umsetzbar sind.

38010-22
Projekt für Aufgabe 5

6 Übernimm die folgende Abbildung in dein Heft und ergänze die fehlenden Begriffe in den blauen Rechtecken. ↪ 3.2



38010-23
interaktive Übung

↪ 3.2

7 Objekte der Klasse `Auto` haben die Attribute `farbe`, `geschwindigkeit` und `hoechstgeschwindigkeit` sowie die Methoden `beschleunigen(int kmh)` und `bremsen(int kmh)`.

a) Zeichne ein Klassendiagramm, das zu dieser Beschreibung passt.
b) Entscheide und begründe, für welche Attribute ein Getter und/oder Setter zur Verfügung gestellt werden sollte.
c) Erstelle ein Struktogramm zur Methode `beschleunigen(int kmh)`. Berücksichtige: Die Höchstgeschwindigkeit darf nicht überschritten werden.

↪ 3.2

8 Entwirf ein Struktogramm für den folgenden Algorithmus von Euklid, der für zwei Zahlen `a` und `b` den größten gemeinsamen Teiler berechnet.
„Solange `a` ungleich `b` ist, wird `b` von `a` abgezogen, falls `a` größer als `b` ist. Ansonsten soll `a` von `b` abgezogen werden."

↪ 3.3

9 In einem gallischen Dorf gibt es Dorfbewohner, Druiden, Barden, Wildscheine, Hunde, Römer, Fische und Häuptlinge. Alle Dorfbewohner haben einen Namen, Muskelkraft und einen Intelligenzquotienten und können einen Zaubertrank trinken. Jeder kann einen Fisch aufheben und einen Fisch werfen. Nur Druiden können einen Zaubertrank brauen. Barden können musizieren und nur ein Häuptling kann die Schildträger rufen. Tiere können einen Laut von sich geben. Das Trinken eines Zaubertrankes erhöht Muskelkraft und IQ.

a) Erstelle ein Klassendiagram, das die oben genannte Situation modelliert. Berücksichtige dabei vorhandene Vererbungshierarchien.
b) Setze das Klassendiagramm in einem neuen Greenfoot-Projekt um. Die Methoden dürfen dabei leer bleiben.
c) Begründe, an welchen Stellen im Programmtext die Annotation `@override` hinzugefügt werden kann.
d) Erstelle eine Objektkarte zu einem Objekt der Klasse Barde.

↪ 3.3

10 Musiker sind sehr vielfältig. Zwar kann jeder Musiker sein Instrument spielen, die Anzahl der Instrumente unterscheidet sich jedoch von Musiker zu Musiker. Dabei haben Schlagzeuger immer nur ein Schlagzeug, Gitarristen aber häufig mehrere Gitarren. Bassisten haben häufig, aber nicht immer genau einen Bass. Auch Sänger sind Musiker. Diese spielen allerdings kein Instrument, sondern können stattdessen singen. Daher ist es auch wichtig, deren Stimmlage zu kennen.

a) Begründe, warum zur Umsetzung eine Vererbungshierachie sinnvoller ist als die Programmierung einer einzigen Klasse `Musiker`.
b) Erstelle ein Klassendiagramm, das die oben genannte Situation modelliert. Berücksichtige dabei vorhandene Vererbungshierarchien.
c) Nenne alle zu überschreibenden Methoden deines Klassendiagramms.
d) Gib den vollen Programmtext des Konstruktors von `Sänger` an.
e) Begründe, ob folgende Beschreibung mit deinem Klassendiagramm erfüllt werden kann: „Viele Gitarristen sind auch Sänger".
f) Begründe, wie die Beschreibung „Schlagzeuger haben nur ein Instrument" im Programm umgesetzt werden kann.
g) Nimm begründet Stellung zu folgender Aussage: „Frauen haben häufig eine höhere Stimmlage, daher sollte man zwei Klassen schreiben. Nämlich einmal die Klasse `Sänger` und einmal die Klasse `Sängerin`!"

11 Visualisiere folgende Variablen wie in Kapitel 3.4 beschrieben, wenn der gespeicherte Wert für diese Variable technisch erlaubt ist. Zeichne ggf. das zugehörige Klassendiagramm. ↪ 3.4

a) `Lehrer hm = new Lehrer("Herr Müller", 39);`
b) `double d1 = 6/2;`
c) `double d2 = 7/2;`
d) `Mensch katze = new Katze("Kali", 'w', 2);`

12* Korrigiere folgende Programmtexte, sodass keine Datentyp-Fehler auftauchen. Sollte der Programmtext bereits korrekt sein, vereinfache ihn, falls möglich. ↪ 3.5

a) `Mensch hm = (Mensch) new Lehrer("Herr Müller");`
b) `boolean f = "false";`
c) `long number = (long) 12345;`
d) `String d = "Das doppelte von 2 ist: " + 2 + 2;`

13* Begründe, warum eine als `String` gespeicherte Zahl nicht mit einer impliziten Typumwandlung in einen `int` umgewandelt werden kann. ↪ 3.5

14* Recherchiere über das „Year 2000 Problem“. Beantworte mit deiner Recherche folgende Fragen. ↪ 3.5

a) Auf welche Art und Weise war die unsinnvolle Benutzung von Datentypen für das Problem verantwortlich?
b) Welche Datentypen hätte man benutzen sollen, um das Problem zu verhindern?
c) Hätte das Problem verhindert werden können, wenn alle Datumsangaben in menschenlesbaren Format als String (z. B. `„21.12.2021 17:00 Uhr“`) gespeichert worden wären? Wäre das eine sinnvolle Lösung gewesen?
d)* Wie wurde das Problem gelöst und war es bei der Lösung nötig, eine Typumwandlung durchzuführen?
e) Wird es in Zukunft vergleichbare Probleme geben?

15* Erkläre die Begriffe Oberklasse, Unterklasse, Vererbung, explizite Typumwandlung, implizite Typumwandlung, Polymorphismus und dynamische Bindung anhand folgender Situation: Es gibt verschiedene Arten von Kinos. Normale Kinos sind von allen Menschen betretbar. VIP-Kinos sind jedoch nur von VIPs betretbar. Zwar sind VIPs auch Menschen, allerdings haben sie einen VIP-Ausweis, der durch seine Ausweisnummer definiert ist. Mit diesem können sie in ein VIP-Kino gehen. Dort wird ein VIP jedoch immer um eine Spende für wohltätige Zwecke gebeten. Um diese zu erfüllen, übergibt die Person die eigene Kreditkarte und schreibt einen Geldbetrag auf einen Zettel. Der Mitarbeiter oder die Mitarbeiterin interpretiert den Text auf dem Zettel als Ganzzahl in Euro und tipp sie ins System ein. Der entsprechende Betrag wird anschließend als Kommazahl in Euro und Cent vom Konto des VIP abgebucht. Geht der VIP in ein normales Kino, wird er dort wie ein Nicht-VIP behandelt. So muss er zum Beispiel wie jede andere Person in einer Schlange stehen. Im Unterschied zu einem Nicht-VIP ist das Warten in der Schlange für einen VIP jedoch weniger lästig, da er in der Zeit von vielen Fans bewundert wird. ↪ 3.6

DAS GROSSE INFO-QUIZ!

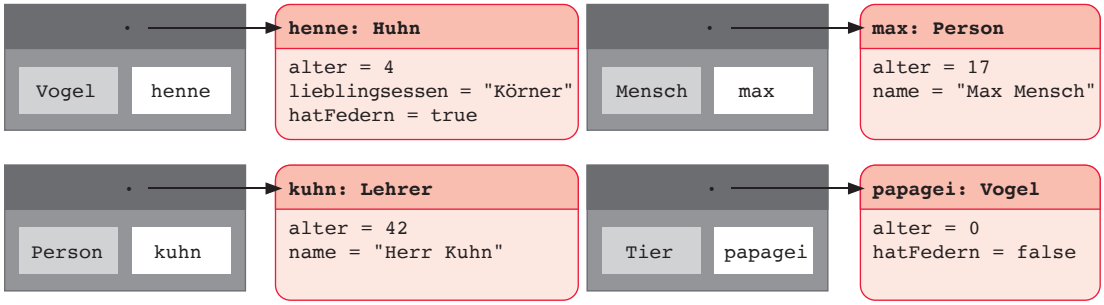
Hier sprechen Begriffe, die du in diesem Kapitel gelernt hast, über sich. Die Zahlen in Klammern geben den Buchstaben im erratenen Begriff an, den du für das Lösungswort brauchst.

A „Ich warne, wenn neue Klassen alte Methoden vergessen." … (7)
B „Ich kann vererbt und nicht mehr ausgetauscht werden." (1)
C „Ich ändere die Darstellung eines Wertes." (1)
D „Bei mir muss man aufpassen, weil ich nicht nur Darstellung, sondern manchmal auch den Wert ändere." (1)
E „Ich gebe einem Methodennamen mehrere Bedeutungen." (13)
F „Ob es so oder anders geht, entscheide am Ende ich." (1)
G „Wenn die Oberklasse speichert und die Unterklasse arbeitet, bin ich nicht weit." (2)
H „Durch mich kann jeder Lehrer auch ein Mensch sein." (1)

38010-19

Hast du das Ziel des Kapitels erreicht? Schätze dich zunächst mithilfe des hinterlegten Bogens selbst ein. Bearbeite dann die Aufgaben und vergleiche mit deiner Selbsteinschätzung.

1 Zeichne ein einziges Klassendiagramm, sodass folgende Variablen erzeugt werden können.



Gib jeweils den Programmtext an, mit dem die Variablen erzeugt werden können.

2 Betrachte das abgebildete Klassendiagramm.

a) Beschreibe für jeden Begriff jeweils, wie das folgende Klassendiagramm erweitert werden muss, damit man den Begriff mit dem Klassendiagramm erklären kann. Formuliere anschließend eine Erklärung.
   I. Vererbungshierarchie
   II. Typprüfung
   III. Überschreiben von Methoden
   IV. Polymorphes Speichern von Objekten
   V.* Dynamische Bindung von Methoden

b) Erweitere das Klassendiagramm mit möglichst wenigen Elementen, sodass mit dem fertigen Diagramm alle Begriffe erklärt werden können.

c) Begründe, wie viele Arten von polymorpher Speicherung von Objekten mit dem Diagramm möglich sind.