Lernziele

Zentrale Konzepte in diesem Kapitel Abstraktion, Objektdiagramme, Modularisierung, Methodenaufrufe, Objekterzeugung, Debugger

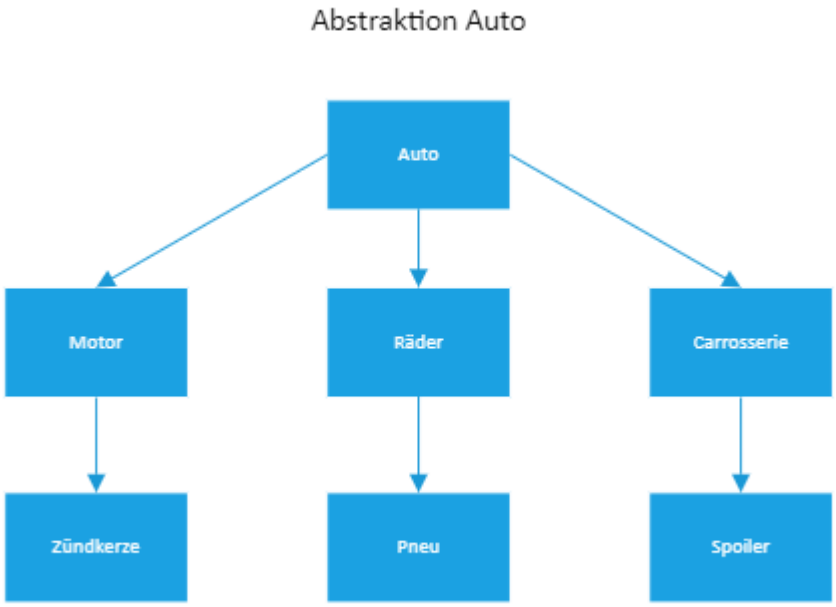
Java-Konstrukte in diesem Kapitel: Klassen als Typen, logische Operatoren (&&, ||), Verkettung von Zeichenketten, Modulo-Operator (%), Objekterzeugung (new), Methodenaufrufe (Punkt-Notation), this

3.1 Abstraktion und Modularisierung

Konzept
Abstraktion ist die Fähigkeit Details von Bestandteilen zu ignorieren, um den Fokus der Betrachtung auf eine höhere Ebene lenken zu können

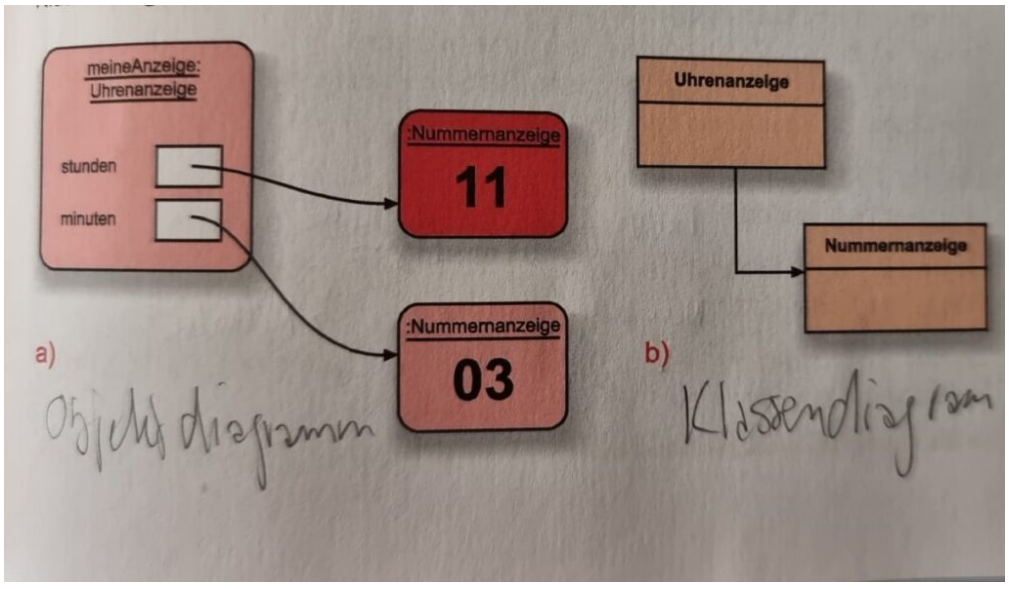
|-> Es wird versucht grosse Aufgaben in verschiedene Teilaufgaben zu zerlegen, diese wieder in Unterteilaufgaben etc., bis die einzelnen Aufgabe klein genug für eine übersichtliche Lösung sind. (Abstraktion) -> Diese Technik ist auch unter der Bezeichnung **Teile und Herrsche** bekannt

Ein Beispiel:



Konzept
Modularisierung ist der Prozess der Zerlegung eines Ganzen in wohldefinierte Teile, die getrennt erstellt und untersucht werden können und die wohldefinierter Weise interagieren

3.2 Klassen- und Objektdiagramme



Konzept
Ein **Klassendiagramm** zeigt die Klassen einer Anwendung und die Beziehungen zwischen diesen Klassen. Es liefert Informationen über den Quelltext. Es präsentiert eine statische Sicht auf ein Programm

Konzept
Ein **Objektdiagramm** zeigt die Objekte und ihre Beziehungen zu einem bestimmten Zeitpunkt während der Ausführung einer Anwendung. Es repräsentiert eine dynamische Sicht auf ein Programm.

Ein sehr wichtiger Fact: Das Objektdiagramm zeigt ein weiteres wichtiges Detail: Wenn eine Variable ein Objekt enthält dann ist dieses Objekt nicht direkt in der Variablen selbst abgelegt sondern die Variable enthält lediglich eine Referenz auf ein Objekt.
In dem Diagramm ist die Variable durch ein weisses Feld symbolisiert und die Referenz durch ein Pfeil. Das Objekt, das referenziert wird, ist ausserhalb des Objekts gespeichert, dass die Referenz hält. Die Objekte sind durch die Objektreferenz miteinander verbunden.

Ein Beispiel:

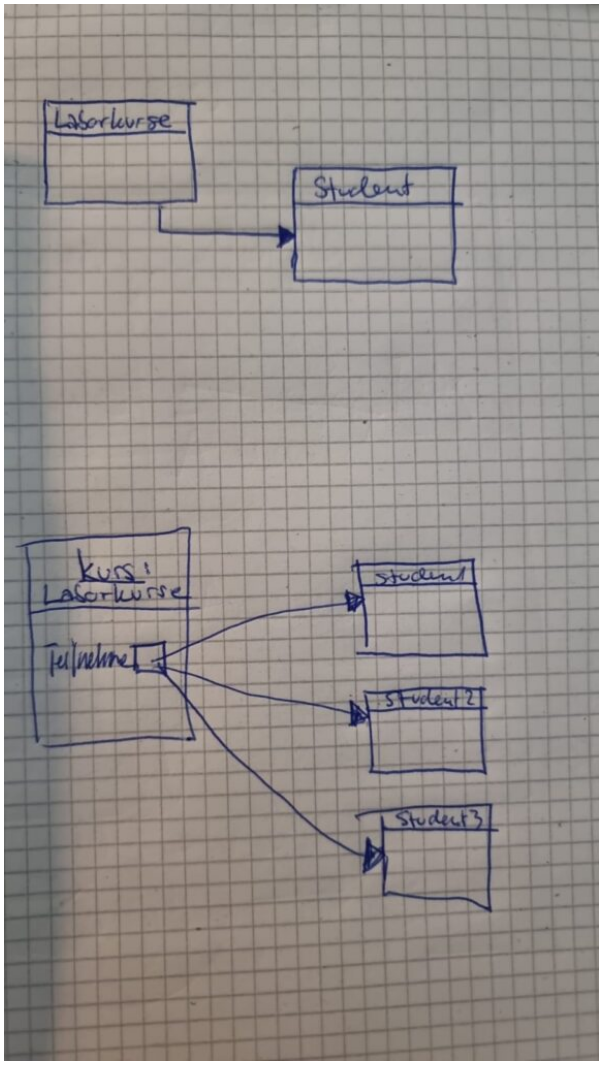
```
NummerAnzeige test = new NummerAnzeige();
test.setNummer(11);

NummerAnzeige test1 = test
test1.setNummer(14);

// Dann hat test auch die Nummer 14 gespeichert
```

Konzept
Objektreferenz: Variablen von Objekttypen speichern Referenzen auf Objekte.

Übung 3.1 Nehmen Sie an, wir erzeugen ein Laborkurs-Objekt und drei Studenten-Objekte. Anschliessend tragen wir die drei Studenten in den Kurs ein. Versuchen Sie ein Klassendiagramm und ein Objektdiagramm für diese Situation zu zeichnen. Zeigen Sie die Unterscheide zwischen den Diagrammen auf und erläutern Sie diese



Übung 3.2 Zu welchen Zeitpunkt kann sich ein Klassendiagramm ändern? Wie wird es geändert?

Bei Veränderung in der Klasse/Quelltext oder beim Hinzufügen einer Klasse

Übung 3.3 Zu welchen Zeitpunkt kann sich ein Objektdiagramm ändern? Wie wird es geändert?

Wenn sich Zustände ändern. z.B. durch Methoden Aufrufen oder Zuweisungen

Übung 3.4 Schreiben Sie eine Definition für ein Datenfeld **tutor**, das eine Referenz auf ein Objekt des Typs **Lehrender** halten kann.

```
private Lehrender tutor;
```

3.3 Primitive Typen und Objekttypen

Konzept
Die Primitiven Typen in Java sind die Typen, die keine Objekttypen sind. Die gebräulichsten primitiven Typen sind **int, boolean, char, double** und **long**. [redacted]

Typname	Beschreibung	Beispielkonstanten		
Ganze Zahlen:				
byte	ganze Zahl in 8 Bit	24	-2	
short	ganze Zahl in 16 Bit	137	-119	
int	ganze Zahl in 32 Bit	5409	-2003	
long	ganze Zahl in 64 Bit	423266353	55L	
Gleitpunktzahlen:				
float	einfache Fließkommazahl (32 Bit)	43.889F		
double	doppelte Fließkommazahl (64 Bit)	42.63	2.4e5	
Andere Typen:				
char	einzelnes Zeichen (16 Bit)	'm'	'?'	'\u00F6'
boolean	ein boolescher Wert (wahr oder falsch)	true	false	

Primitive Typen

Typ	Minimum	Maximum
byte	-128	127
short	-32768	32767
int	-2147483648	2147483647
long	-9223372036854775808	9223372036854775807
	Positives Minimum	Positives Maximum
float	1.4e-45	3.4028235e38
double	4.9e-324	1.7976931348623157e308

3.4 Die logischen Operatoren

Logische Operatoren arbeiten mit booleschen Werten (wahr und falsch bzw. **true** und **false**) und liefern als Ergebnis wieder einen booleschen Wert. Die drei wichtigsten logischen Operatoren sind **und**, **oder** und **nicht**. In Java werden diese folgendermaßen notiert.

&& (und)
|| (oder)
! (nicht)

Der Ausdruck **a && b**
ist dann wahr (liefert **true**) , wenn sowohl **a als auch b wahr sind**, in allen anderen Fällen ist er falsch (lieferet **false**)

Der Ausdruck **a || b**
liefert true, wenn **entweder a oder b oder beide true liefern**, und false, wenn beide **false** liefern.

Der Ausdruck **!a** liefert true, **wenn a false ist** und fals wen a true ist

```
public void setzeWert(int ersatzwert)
{
  if((ersatzwert >= 0 ) && (ersatzwert < limit))
  {
    wert = ersatzwert;
  }
}
```

Übung 3.10 Was passiert, wenn Sie die Methode setzeWert mit einem ungültigen Wert aufrufen? Ist das eine gute Lösung? Können Sie sich eine bessere vorstellen?

1. Es passiert nicht
2. Nein es ist keine gute Lösung
3. Es sollte mindestens eine Fehlermeldung ausgegeben werden.

Übung 3.11 Was würde passieren, wenn Sie den Operator >= in der Prüfung durch >

ersetzen würden, und zwar in folgender Weise: if((ersatzwert > 0) && (ersatzwert < limit))

Die 0 wäre nicht mehr zulässig

Übung 3.12 Was würde passieren, wenn Sie den Operator && in der Prüfung durch || ersetzen würden, und zwar in folgender Weise: if((ersatzwert >= 0) || (ersatzwert < limit))

Der Test würde das Ergebnis true liefern, fall eine der Bedingung true liefert. Die Lösung wäre immer true

Übung 3.13 Welche der folgende Ausdrücke liefern tue?

!(4<5) -> false
!false -> true
(2>2) || ((4 ==4) && (1<0)) -> false
(2>2) || (4 ==4) && (1<0) -> false
(34 != 33) && !false -> true

Übung 3.14 Schreiben Sie einen Ausruck mit zwei booleschen Variablen a und b, der tue liefert, wenn entweder a und b beide true sind oder beide false sind

(a == true && b == true) || (a == false && b == false) => **a == b**
zwischen Schritt wäre (a && b) || (!a && !b)

Übung 3.13 Schreiben Sie einen Ausdruck mit zwei booleschen Variablen a und b der true liefert, wenn nur **genau** eine von **beiden true** ist, und der false liefert, wenn **a und b beide false oder beide true** sind

(a == true && b == false) || (a == false && b == true) => a != b
zwischen Schritt wäre (a && !b) || (!a && b)

Übung 3.13 Betrachten Sie den Ausdruck (a && b). Schreiben Sie einen Äquivalenten Ausdruck (einen, der in exakt den gleichen Fällen für die gleichen Werte true liefert), ohne den Operator && zu benutzen

a && b => a == true && b == true
!a => false, !b => false
!a || !b => a == false || b == false

Lösung: !(!a || !b)

3.4 Verkettung

Wichtig beim + – Operator:
Wenn am anfang eine Zahl steht z.B. 0 + 2 => dann wird es addiert
Wenn am Anfang ein String steht. z.B. “0” + 3 -> dann wird es verkettet “02”

3.5 Der Modulo-Operator %

```
public void erhoehen(){
  wert = (wert + 1) & limit;
}
```

Der Modulo-Operator berechnet den Rest einer ganzzahligen Division. Beispielsweise kann das Ergebnis der Division 27/ 4 durch 2 ganze Zahlen ausgedrücket werden: Ergebnis 6, Rest 3

Der Modulo liefert lediglich den Rest einer solchen Division: 3

3.6 Objekt erzeugen Objekte

Konzept
Objekterzeugung: Objekte können andere Objekte mit dem **new** Operator erzeugen

```
Objekt1 obj1 = new Objekt1();
```

3.7 Mehrere Konstruktoren

Konzept
Überladen: Eine Klasse kann mehr als einen Konstruktor oder mehr als eine Methode mit dem gleichen Namen enthalten, solange jede von Ihnen einen unterscheidbaren Satz von Parametertypen definiert.

```
new Uhrenanzeige();
new Uhrenanzeige(stunde, minute);
```

3.8 Methodenaufrufe

Konzept
Methoden können andere Methoden der eigenen Klasse als Teil ihrer Implementierung aufrufen. Dies wird als **interner Methodenausruf** bezeichnet.

Aufruf -> anzeigeAktualisieren()
Methodensignatur: private void anzeigeAktualisieren();

Struktur: Methodenname (Parameterliste)

Konzept
Methoden können Methoden von anderen Objekten über die Punkt-Notation aufrufen. Dies wird als externer Methodenaufruf bezeichnet.

Struktur: Objekt. Methodenaufruf (Parameterliste)

3.9 Debugger

Konzept
Ein Debugger ist ein Softwarewerkzeug, mit dessen Hilfe die Ausführung eines Programms untersucht werden lann. Es kann benutzt werden, um Fehler (**Bugs)** zu finden

Zusammenfassung der Konzepte

- **Abstraktion** Abstraktion ist die Fähigkeit, Details von Bestandteilen zu ignorieren, um den Fokus der Betrachtung auf eine höhere Ebene zu lenken zu können.
- **Modularisierung** ist der Prozess der Zerlegung eines Ganzen in wohldefinierte Teile, die getrennt erstellt und untersucht werden können und die in wohldefinierter Weise interagieren
- **Klassen** definieren Typen. Ein Klassenname kann als Typname in einer Variablendeklaration verwendet werden. Variablen, die als Typ eine Klasse haben, können Objekte dieser Klasse halten.
- **Klassendiagram** zeigt die Klassen einer Anwendung und die Beziehung zwiscehn

diesen Klassen, Es liefert Informationen über den Quelltext. Es repräsentiert eine statische Sicht auf ein Programm.

- **Objektdiagram** zeigt Objekte und ihre Beziehung zu einem bestimmten Zeitpunkt während der Ausführung einer Anwendung. Es präsentiert eine dynamische Sicht auf ein Programm.
- **Objektreferenz** Variablen von Objekttypen speichern Referenz auf Objekte.
- **Primitiver Typen** in Java sin die Typen , die keine Objekttypen sind. Die gebräuchlisten primitiven Typen sind **int, boolean, char, double und long**. **Primitive Typen haben keine Methode**
- **Objekterzeugung** Objekte können andere Objekte mit dem **new** Operator erzeugen.
- **Überladen** Eine Klasse kann mehr als einen Konstruktor oder mehr als eine Methode mit dem gleichen Namen enthalten, solange jede von ihnen eine unterscheidbaren Satz von Parametertypen definiert.