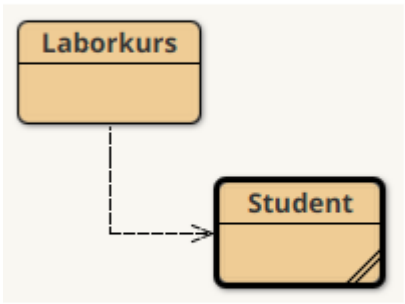
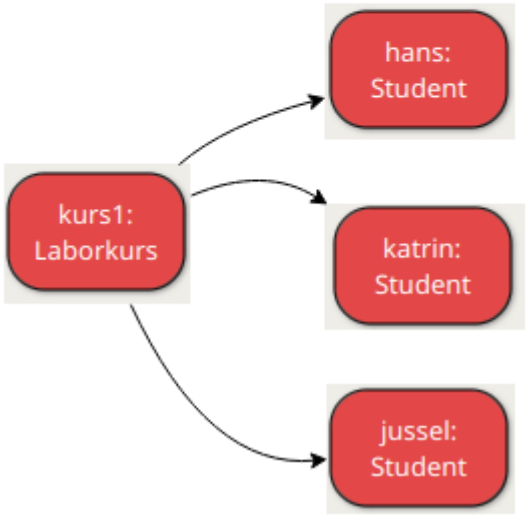
Lösungsvorschläge zum Kapitel 3

Übung 3.1

Klassendiagramm:
Die Klasse "Laborkurs" verwaltet Objekte der Klasse "Student".

Objektdiagramm:
Das Objekt "Kurs1" vom Typ "Laborkurs" zeigt auf 3 Objekte des Typs "Student", und zwar "hans", "katrin" und "jussel"





Übung 3.2

Ein **Klassendiagramm** ändert sich, wenn Sie den Quelltext modifizieren. Dies geschieht, indem die Beziehungen zwischen den Klassen verändert werden oder Klassen neu erstellt bzw. gelöscht werden.

Übung 3.3

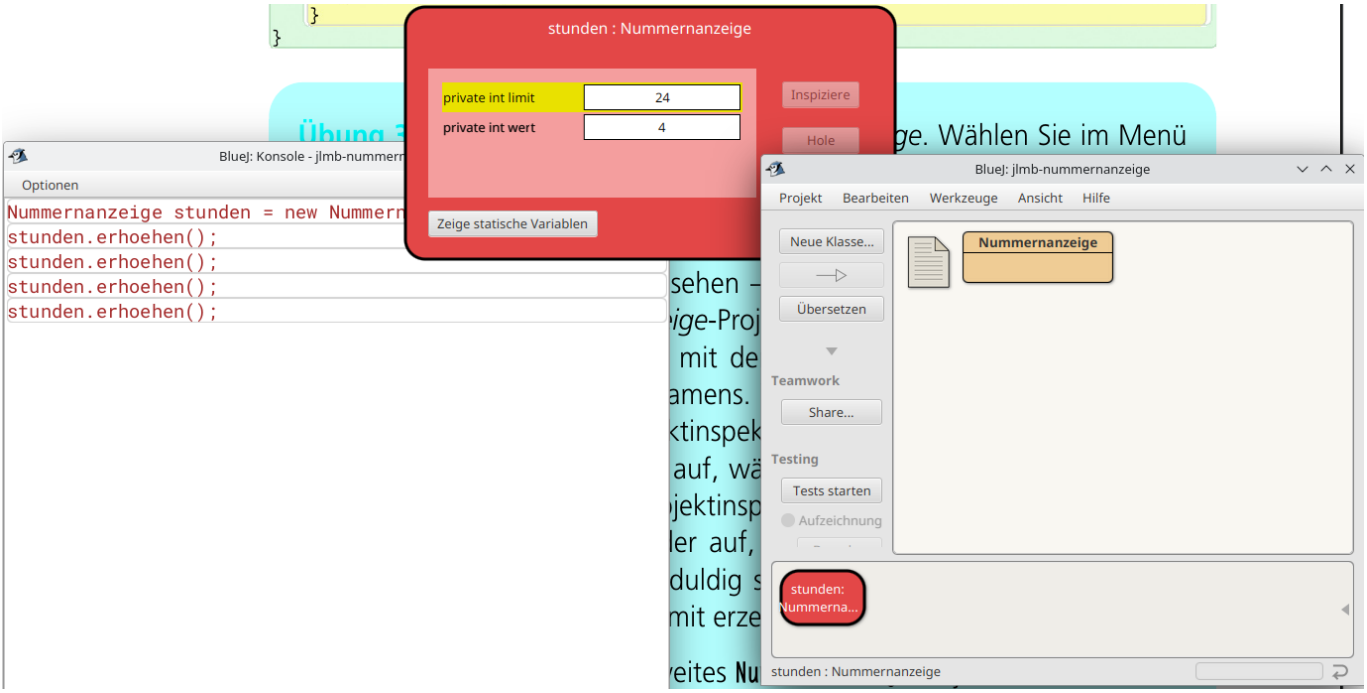
Ein **Objektdiagramm** ändert sich, während das Programm läuft. Es kann sich durch das Erstellen neuer Objekte oder den Aufruf von Methoden verändern.

Übung 3.4

Jede Klassendefinition erzeugt in Java einen Variablentyp für Objekvariablen - die Klasse "Lehrender" kann also bei der Deklaration von Variablen verwendet werden wie beispielsweise `int`, die so deklarierte Variable kann dann auf Objekte des Typs "Lehrender" verweisen:

```
private Lehrender tutor;
```

Übung 3.5



Übung 3.6

Wenn der wert für die Anzeige der Minuten auf 0 zurückspringt, ist eine Stunde vergangen und die `erhoehen()`-Methode der Stunden muss aufgerufen werden. man muss also jedes Mal prüfen, ob der Wert der Minuten auf Null springt, und wenn ja die Stunden erhöhen.

Übung 3.8

```
Nummernanzeige.gibWert();
Error: Methode gibWert() ist nicht statisch und kann nicht aus einem
statischen Kontext referenziert werden
```

Bei normalen Klassen und Methode muss man zuerst ein Objekt der Klasse erzeugen und kann erst anschließend die Methode dieses Objekts aufrufen. Der Versuch, die Methode "generisch" auf der Klasse aufzurufen schlägt fehl. Wenn man Methoden benötigt, die man verwenden kann, ohne zuvor ein Objekt zu erstellen muss man "statische" Klassen und Methoden verwenden - die Fehlermeldung deutet dies an.

Übung 3.9

```
na.setzeWert(int 5);
Error: ".class" erwartet
```

Der Fehler liegt darin, dass `setzeWert` zwar einen Parameter des Typs `int` erwartet, das darf man

aber beim Aufruf der Methode nicht mit übergeben, es muss als Parameter eben einfach eine ganze Zahl angegeben werden. Die Fehlermeldung hilft hier nicht so richtig weiter.

Übung 3.10

- Nichts passiert.
- Nein, denn man kann so nicht erkennen, dass was schief gegangen ist.
- Es sollte eine Fehlermeldung ausgegeben werden.

Übung 3.11

Man könnte für die Anzeige den Wert `0` nicht mehr setzen.

Übung 3.12

Der Test würde das Ergebnis `true` liefern, falls mindestens eine der Bedingungen true ist, solange `limit` größer oder gleich Null ist, wäre der Test also immer `true`.

Übung 3.13

- ! (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

Der Ausdruck a==b erfüllt die geforderten Bedingungen:

a	**b**	**a==b**
true	true	true
true	false	false
false	true	false
false	false	true

Übung 3.15

Der Ausdruck `( a || b ) && !( a && b)` erfüllt die geforderten Bedingungen.

a	**b**	**(a || b) && !(a && b)**
true	true	false
true	false	true
false	true	true

a	b	(a || b) && !(a && b)
false	false	false

Übung 3.16

Der Ausdruck `!(!a || !b)` erfüllt die geforderten Bedingungen.

a	b	!(!a || !b)
true	true	true
true	false	false
false	true	false
false	false	false

Übung 3.17

Das klappt nicht für alle Ausgaben, die Methode unterstellt, dass der Wert nur zwei Stellen hat und füllt diese mit führenden Nullen auf, wenn nötig.

Übung 3.18

Nein, beide Darstellungen funktionieren. Die Verkettung mit dem leeren String stellt hier nur sicher, dass das Ergebnis der Methode vom Typ `String` ist, obwohl `wert` vom Typ `int`, also eine Zahl ist.

Übung 3.19

```
9 + 3 + "See"
"12See"   (String)
"See" + 9 + 3
"See93"   (String)
```

Im ersten Fall werden zunächst die Zahlen 9 + 3 addiert, das Plus-Zeichen steht hier also (noch) für die Rechenoperation - dann komm die Verkettung mit dem String, das Ergebnis 12 wird mit dem String verkettet, man erhält den String "12See".

Im zweiten Fall sorgt der Srring zu Beginn der Operation dafür, dass alle weiteren "Plus"-Zeichen als Verkettungsoperator betrachtet werden, weil von Anfang an klar ist, dass das Ergebnis ein String sein wird.

Übung 3.20

Die "Modulo"-Operation gibt den Rest bei der Divison zweier ganzer Zahlen zurück:

```
7 mod 5 = 2 //  7 = 1 * 5 + 2 (5 geht einmal ganz in 7, dann bleibt ein Rest
```

```
von 2
167 mod 10 = 7  // 167 = 16 * 10 + 7 ("167/10 = 16 Rest 7")
```

Java hat dafür den Modulo Operator %.

Übung 3.26

```
public void erhoehen() {
  wert = wert + 1;
  if(wert >= limit) {
    wert = 0;
  }
}
```

Beide Wege sind gleich gut - man sollte den nehmen, den man als Programmierer besser versteht.