

Vortrag zum Thema "Einführung in die Objektorientierung"

Wiederholung:

Bisher haben wir Daten anhand feste Typunterscheidungen angelegt. Wir mussten zwischen Ganzzahl, Kommazahl, Wahrheitswert, Zeichenkette, etc... unterscheiden und waren darin fest vorgegeben.

Beispiel:

```
int antwort = 42;
```

Bedeutung: Im Arbeitsspeicher wird Platz für eine Ganzzahl (int) reserviert. Diesen Platz nennen wir für die spätere Verwendung "antwort" und schreiben die Zahl 42 an diesen Platz.

Frage:

Was können wir machen wenn wir Daten haben, die sich nicht einfach in das Muster Ganzzahl ODER Zeichenkette ODER Wahrheitswert ODER ... einordnen lassen?

Beispiel:

Wir wollen für einen Onlineshop die dortigen Artikel speichern. Zu einem Artikel gehört ein Name, die Anzahl wie oft er noch auf Lager ist und der Preis

Name	Anzahl	Preis
Fernseher	10	399,95
Mikrowelle	5	59,50
Kühlschrank	3	280,00
...

Problem:

Offenbar muss so ein Artikel einen String haben, sonst kann man den Namen nicht speichern. Aber so ein Artikel muss auch ein int und ein float haben um Anzahl und Preis festzuhalten.

Wie sorgt man nun dafür, dass Name, Anzahl und Preis eine Einheit bilden? Ich möchte, dass zu einem Artikel diese drei Dinge immer beieinander sind.

Selbes Problem bei

- Bankkunden (Name, Kontostand, darf_überziehen?, ...)
- Fenster einer graphischen Benutzeroberfläche (Größe, Titel, ...)
- mp3-Sammlung (Titel, Interpret, Länge, Album, ...)
- ...

Kurz: Bei so ziemlich allen Dingen des realen Lebens.

Lösungsidee:

Wir müssen uns für so einen Artikel im Onlineshop einen eigenen Datentyp bauen! Also einen Datentyp, der sowohl String als auch int als auch float kann. (Analog für Bankkunden, Fenster, ...)

Wie setzt man das in Java um?

==> Codebeispiel 1

Frage:

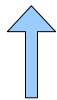
Und was ist nun ein Objekt?

Antwort:

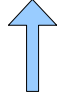
Ein Objekt ist die Instanz einer Klasse.

Beispiel:

```
Artikel beispiel = new Artikel();
```



Klasse



Instanz der Klasse (also das Objekt)

Das Objekt "beispiel" ist also vom (Daten-)Typ "Artikel"

Nochmal im Detail:

```
Artikel beispiel = new Artikel();
```

Folgendes passiert: Es wird Platz für einen Artikel im Speicher reserviert. Hinter der Bezeichnung "beispiel" verbirgt sich dann eine Referenz auf diesen neuen Artikel, nicht der Artikel selbst! Mit anderen Worten: In "beispiel" steht NICHT "name", "anzahl" und "preis" sondern lediglich wo diese zu finden sind.

Was bedeutet das?

Wenn "beispiel" nun kopiert wird, etwa mit

```
Artikel beispiel2 = beispiel;
```

so stehen sowohl in "beispiel" als auch in "beispiel2" lediglich die SELBEN Referenzen, wo die Einträge "name", "anzahl" und "preis" zu finden sind. Ändert man nun mit

```
beispiel.anzahl = 5;
```

die Anzahl, so wird diese auch bei "beispiel2" 5 sein.

--> Codebeispiel 2

Man nennt dies "Seiteneffekt".

Frage:

Wo kommen diese Seiteneffekte besonders zum tragen?

Antwort:

Bei Methodenaufrufe.

Wichtig:

Seiteneffekte gibt es nur bei Objekten (erkennbar an der Großschreibung: "Artikel", "Scanner", ...) nicht jedoch bei primitiven Datentypen (Kleinschreibung: int, float, double, boolean, ...)

--> Codebeispiel 3

War das schon alles?

Nein. Objektorientierung ist mehr, als nur Datentypen zu basteln.

Zu den Dingen die wir in Java abbilden wollen gehören ja nicht nur Daten. Auf diesen Daten werden auch Operationen gemacht, die ganz individuell zu diesen Dingen gehören.

Beispiel:

Während der Lagerbestand eines Artikels - also die Anzahl - niemals kleiner als Null sein darf, wäre das mit dem Kontostand eines Bankkunden durchaus möglich. Dieser wiederum sollte aber nur abheben dürfen, wenn er nicht über sein Limit hinausgeht.

--> Codebeispiel 4

Frage:

Warum steht in den Methoden der Klassen nicht mehr STATIC?

Antwort:

Weil diese Methoden auf den individuellen Daten eines Objektes der Klasse arbeiten.

Die Methode `set_anzahl(int neue_anzahl)` arbeitet auf derjenigen Anzahl die von dem Objekt kommt, welches die Methode aufgerufen hat

```
Artikel art1 = new Artikel();  
Artikel art2 = new Artikel();  
  
art1.set_anzahl(42);  
art2.set_anzahl(23);
```

Nun ist die Anzahl von art1 42 und die von art2 23.

Das "static" ist weggefallen, da nicht auf festen Daten sondern immer auf den Daten des jeweiligen Aufrufers gearbeitet wird.

Wichtige Hinweise zum Schluss:

- 1.) Eclipse-Benutzer müssen für jede Klasse eine eigene Datei anlegen
- 2.) JCreator-Benutzer können mehrere Klassen in eine Datei packen, allerdings darf nur eine Klasse davon public sein.

Fragen?