ÜBUNGS-BLOCK 1
LÖSUNGEN

Aufgabe 1:

Wenn dich jemand fragen würde, was er sich im Bezug auf die Programmierung unter einer Klasse vorstellen kann, was würdest du ihm sagen? Aus welchen Bestandteilen besteht eine Klasse in Java (in der Regel), und was kann man sich unter diesen Bestandteilen vorstellen?

Lösung:

Der Begriff Klasse kommt aus dem Bereich der Objekt-orientierten Programmierung (OOP). Eine Klasse ist in OO-Sprachen eine Datenstruktur zur Beschreibung von Objekten. Insbesondere ist eine Klasse aber kein Objekt, und erst Recht kein Programm. Man kann sich eine Klasse vorstellen wie ein Handbuch zu einem Produkt: Es beschreibt das Produkt nur, aber es ist nicht das Produkt selber.

Eine Klasse besteht in Java meist aus:

1. **Instanz-Variablen (auch Attribute, Member genannt)**
   Das sind die beschreibenden Daten über ein Objekt, also seine Eigenschaften.
2. **Instanz-Methoden**
   Das sind die Dinge, die man mit einem Objekt dieser Klasse tun kann, oder welche es von sich aus tun kann. Also jegliche Aktionen und Interkationen des Objekts oder mit dem Objekt. Bei der Ausführung solcher Methoden bezieht man sich eigentlich immer auf die Instanzvariablen, entweder um deren Werte abzufragen, oder deren Werte zu manipulieren.
3. **Konstruktoren (in vielen Fällen nur einer)**
   Ein Konstruktor kann zur Laufzeit des Programms ein Objekt anhand der Klassenbeschreibung erstellen. Erst dann werden im Speicher wirklich Daten für die Instanz-Variablen angelegt, und erst dann kann man Instanz-Methoden aufrufen.

Allerdings ist keiner dieser Bestandteile Pflicht. Folgender Code definiert schon eine gültige Java-Klasse:

```
class Class {}
```

So etwas macht natürlich keinen Sinn, aber es gibt in der Tat Klassen, die nur manche Dinge aus der obigen Liste implementieren. Man sollte solche Klassen aber nicht als Klassen im eigentlichen Sinne des OOP-Paradigmas ansehen, da das mit Objekten meist nichts mehr zu tun hat. Java „zweckentfremdet“ das Konstrukt der Klassen einfach nur für gewisse andere Strukturen. Wir werden dazu im Laufe des Kurses noch mehr erfahren.

Aufgabe N-1:

Der folgende Code enthält einige Fehler und lässt sich deshalb nicht kompilieren. Finde diese Fehler und berichtige sie.

Anmerkung: Natürlich sollst du diesen Code nicht in deine IDE hineinkopieren, um dir die Fehler dort anzeigen zu lassen. Du sollst die Fehler selbständig auffinden.

```
class Kunde {

    String vorname, nachname;
    int kundenNummer;

    void nummerAendern(neueNummer){
       kundenNummer = neueNummer;
    }

    Konstruktor(String derVorname & String derNachname & int dieKundenNummer) {
       vorname = derVorname;
       nachname = derNachname;
       kundenNummer = dieKundenNummer;
    }
}
```

Lösung:

Der erste Fehler befindet sich in der Parameterliste der "nummerAendern"-Methode:

```
void nummerAendern(neueNummer)
```

Der Fehler liegt darin, dass für den Parameter "neueNummer" kein Datentyp angegeben ist. Denke daran, dass Java streng typisiert ist. Für jede Variablen-Deklaration muss damit nicht nur ein Name, sondern auch ein Datentyp angegeben werden. Richtig wäre:

```
void nummerAendern(int neueNummer)
```

Der zweite Fehler liegt im Namen des Konstruktors. Der Konstruktor-Name muss immer mit dem Klassen-Namen übereinstimmen. Statt:

```
Konstruktor(...)
```

muss es in diesem Fall also heißen:

```
Kunde(...)
```

Als Letztes ist die Parameterliste des Konstruktors syntaktisch falsch. Mehrere Parameter sind zwar möglich, allerdings werden sie nicht mit einem "&"-Zeichen verknüpft, sondern statt dessen mit einem Komma getrennt. Richtig wäre also:

```
Kunde (String derVorname, String derNachname, int dieKundenNummer)
```

Zuletzt noch eine Anmerkung:
Du hast vielleicht gedacht, dass folgende Zeile ungültig ist:

```
String vorname, nachname;
```

Das wäre verständlich, denn diese Art der Instanz-Variablen-Deklaration habe ich in den Videos nicht erwähnt.

Tatsache ist jedoch, dass diese Zeile durchaus korrekt ist! Falls man mehrere Variablen des selben Typs deklarieren will, so kann man das durch diese Syntax realisieren. Man hängt hinter die Deklaration der ersten Variablen dazu also einfach ein Komma an, und listet dann die Namen von weiteren Variablen - alle durch Komma getrennt. So könnte man z.B. auch vier int-Variablen wie folgt deklarieren:

```
int a, b, c, d;
```

Man gibt also für alle weiteren Variablen lediglich den Namen an, nicht aber einen Typen. Alle Variablen bekommen dann den selben Typ wie die erste Variablen, in diesem Fall `int`.

Achtung: Du weisst, dass man im Zuge der Deklaration auch eine Zuweisung machen kann. In diesem Falle ist es wichtig zu verstehen, dass die Zuweisung lediglich für die am nächsten links stehende Variable durchgeführt wird, aber nicht für alle Variablen in der Deklaration!

```
int a, b, c, d = 10;
```

Hier wird der Wert 10 also nur der Variablen "d" zugewiesen, NICHT aber auch den Variablen a, b und c. Eine Zuweisung muss nach wie vor für jede Variable einzeln vorgenommen werden, z.B.:

```
int a = 1, b = 2, c = 3, d = 4;
```

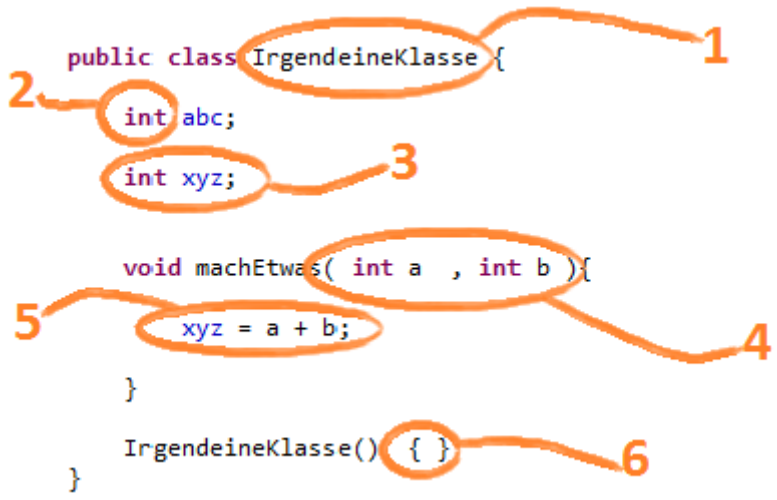
...Warum habe ich diese Möglichkeit der Deklaration mehrerer Variablen in einem Schritt nicht in den Videos erwähnt? Ganz einfach: Du solltest das vermeiden! Üblicherweise werden Variablen nämlich zeilenweise deklariert, und man rechnet nicht damit, dass mehrere Deklaration in einem Schritt passieren. Es kann in so einem Fall passieren, dass jemand die Deklaration der anderen Variablen überliest. Außerdem ist es fehleranfällig, wenn man das ganze dann noch mit Zuweisungen kombiniert.

Ich wollte mit dieser Aufgabe also darauf aufmerksam machen, dass es durchaus geht, damit du dich nicht wunderst, wenn du so etwas irgendwo siehst. Aber ich würde dir davon abraten, diese Kurzschreibweise zu benutzen! Es wäre also schöner gewesen, wenn der Code in der Klasse "Kunde" wie folgt ausgesehen hätte:

```
String vorname;
String nachname;
int kundenNummer;
```

Aufgabe N-2:

Benenne die markierten Teile des folgenden Codes nach ihren offiziellen Bezeichnungen:



Lösung:

1: Klassen-Name

2: Datentyp int (für "integer", das englische Wort für "ganze Zahl")

3: (Instanz-)Variablen-Deklaration einer Variablen mit Namen "xyz" und Datentyp int

4: Parameterliste (manchmal auch: Argumenten-Liste)

5: (komplexe) Anweisung, bestehend aus Addition und Zuweisung

6: (leerer) Konstruktor-Rumpf (allgemeiner: Code-Block)

Aufgabe N-3:

Gegeben sei die Klasse "Kunde" aus Aufgabe N-1, allerdings gehen wir für diese Aufgabe von einer fehlerfreien Version dieser Klasse aus. Du kannst für diese Aufgabe also entweder deine eigene Lösung der Aufgabe N-1 verwenden, oder auf die Musterlösung zurückgreifen.

Deine Aufgabe:
Implementiere *eine* Methode in dieser Klasse, mit der sich sowohl der Vorname als auch der Nachname eines Kunden ändern lässt.

Lösung:

```
void nameAendern(String neuerVorname, String neuerNachname){
  vorname = neuerVorname;
  nachname = neuerNachname;
}
```

Der Name der Methode ist natürlich frei wählbar. Wichtig ist, dass die Namenskonvention eingehalten wird (lower case camel case), und keine Umlaute vorhanden sind.

Die Reihenfolge der Parameter ist in diesem Fall auch egal, genauso wie die Reihenfolge der Zuweisungen. Folgendes wäre also genau so richtig:

```
void nameAendern(String neuerNachname, String neuerVorname){
  nachname = neuerNachname;
  vorname = neuerVorname;
}
```

oder:

```
void nameAendern(String neuerNachname, String neuerVorname){
  vorname = neuerVorname;
  nachname = neuerNachname;
}
```

Die Namen der Parameter sind grundsätzlich auch frei wählbar - man sollte nur darauf achten, dass man einen Namen wählt, der Sinn macht, und nicht dem Namen der Instanz-Variablen gleicht! Folgendes wäre also problematisch:

```
void nameAendern(String vorname, String nachname){
  vorname = vorname;
  nachname = nachname;
}
```

Dazu wirst du aber zu einem späteren Zeitpunkt des Kurses noch mehr erfahren - merke dir für den Moment einfach, dass die Namen von Parametern sich nicht mit den Namen von Instanz-Variablen gleichen sollten.

Wichtig ist auch, dass die Zuweisungen zu den Instanz-Variablen geschehen, d.h. dass die Instanz-Variablen links vom "="-Zeichen stehen und die Parameter rechts davon, und nicht andersherum.

Aufgabe 2:

Ein Obsthändler möchte für seinen Warenbestand ab sofort eine Software einsetzen, um mit dem Geist der Zeit zu gehen. Im Mittelpunkt der Software müssen entsprechend Daten stehen, die ein Obst beschreiben.

Implementiere diese Daten in einer geeigneten Klasse, anhand folgender Richtlinien:

- Jedes Obst hat eine Bezeichnung, zB „Apfel“, „Birne“ usw.
- Jedes Obst hat einen Einkaufspreis (in Euro)...
- ...sowie einen Verkaufspreis. Der Händler setzt den Verkaufspreis gemäß des Einkaufspreises immer so an, dass er beim Verkauf 100% Gewinn macht
- Zudem soll die Software zu jedem Obst eine Nährwert-Angabe enthalten (in kcal)

Um besser konkurrieren zu können, hat sich der Obsthändler zudem entschieden seine Ware etwas „aufzupeppen“. Er hat sich dafür eine neue Wunder-Chemikalie besorgt, die er mit Hilfe einer Spritze in sein Obst einführen kann. Die Chemikalie bewirkt, dass das Obst schöner glänzt und saftigere Farben hat, was dazu führt, dass der Obsthändler seine Waren nochmal doppelt so teuer verkaufen kann.

Zudem steigt der Nährwert des Obstes pauschal um 50 kcal.

Dieser „Aufpeppen“-Vorgang muss auch in der Software implementiert sein, damit sie aufgepepptes Obst entsprechend widerspiegeln kann.

Lösung:

```
class Obst {

	String bezeichnung;
	int einkaufspreis; // in EUR Cent
	int verkaufspreis; // in EUR Cent
	int naehrwert; // in kcal

	Obst(String dieBezeichnung, int derEinkaufspreis, int derNaehrwert) {
		bezeichnung = dieBezeichnung;
		einkaufspreis = derEinkaufspreis;
		verkaufspreis = einkaufspreis + einkaufspreis;
		naehrwert = derNaehrwert;
	}

	void aufpeppen() {
		verkaufspreis = verkaufspreis + verkaufspreis;
		naehrwert = naehrwert + 50;
	}
}
```

Kommentar zur Lösung:

Rein syntaktisch gesehen ist die Reihenfolge von der Definition der einzelnen Bestandteile egal. So kann beispielsweise der Konstruktor erst hinter der Methode definiert werden. Genauso könnten die ersten beiden Instanzvariablen oben definiert werden, gefolgt von der Methode, dem Konstruktor, und zuletzt den beiden restlichen Instanzvariablen. Allerdings sollte man es übersichtlich halten, und diese Dinge nicht so sehr durchmischen. Üblicherweise werden Instanzvariablen zu Beginn definiert. Ob man dann zuerst die Konstruktoren oder Methoden definiert ist jedem selbst überlassen. Man sollte es aber bei allen Klassen gleich machen.

Was in der Lösung vorallem erwartet wird:

- Korrekte Namenskonventionen
- Die aufpeppen() Methode mit korrekter Logik. Wenn die Methode einen anderen Namen hat, ist es nicht schlimm, solange der Name Sinn macht und mit einem kleinen Buchstaben beginnt. Natürlich ist es hier auch egal, ob man zuerst den Verkaufspreis und dann den Nährwert erhöht, oder anders herum.
- Kommentare für die Instanz-Variablen. Es kann relevant sein ob die Preise in Euro oder Dollar sind, wenn man etwas umrechnen will. Genauso ist die Angabe eines Nährwert-Attributs ohne Kommentar, um welche Einheit es sich handelt, zu ungenau. Immer daran denken: Man sollte möglichst so programmieren, dass ein anderer den Code gut versteht. Das kommt einem selbst auch zu Gute!
- Der Konstruktor mit den drei Paramtern...

... die Reihenfolge der Parameter ist egal, generell auch die Reihenfolge der Zuweisungen im Rumpf – in dieser konkreten Lösung ist es aber wichtig, dass die Zuweisung zum Verkaufspreis erst nach der Zuweisung zum Einkaufspreis erfolgt. Sonst würde noch der Default-Wert des Einkaufspreises genommen (0). Wenn man aber bei der Zuweisung zum Verkaufspreis statt „einkaufspreis“ die Variable „derEinkaufspreis“ aus der Parameterliste nutzen würde, wäre es dahingegen nicht wichtig, ob man den Verkaufspreis vor oder nach dem Einkaufspreis zuweist.

Wichtig ist beim Konstruktor vorallem, die richtige Design-Entscheidung zu treffen: Es sollte keinen Parameter für den Verkaufspreis geben! Denn dann müsste man beim Aufruf, also bei der Instanz-Erzeugung, selber sicherstellen, dass man für den Verkaufspreis das doppelte des Einkaufspreises angibt. Da das aber garantiert sein soll, sollte der Konstruktor diese Regel selbstständig anwenden, und die Entscheidung nicht „nach außen hin“ anbieten.