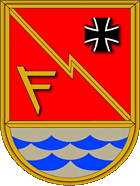
,

Schule Informationstechnik

der Bundeswehr



**Sprachausbildung Java**



**Übungen**

**"Objektorientierte Programmierung"**

Klassen, Objekte, Attribute und Methoden

# Verständnis

Wir beginnen zunächst mit einer Reihe von Verständnisfragen. Vervollständigen Sie folgende Aussagen bzw. geben Sie an, welche der unten stehenden Aussagen korrekt ist:

1. Um den kontrollierten Zugriff auf Datenelemente einer Klasse sicherzustellen, werden die Datenelemente (Attribute) normalerweise als private deklariert und durch \_getter / setter\_\_ gelesen oder geändert.
2. Gegeben sei der folgende Ausschnitt einer Klassendefinition:

class Demo { double zahl; /\* … \*/};

* 1. Ist das Datenelement zahl dann als public deklariert? Nein, package.

1. Elemente von verschiedenen Klassen
   1. ~~müssen unterschiedliche Namen haben~~.
   2. können den gleichen Namen haben.
2. Gegeben sei folgende Klassendefinition:

class Test

{

int iCount;

public void setCount(int iN){

} // iN -> iCount

};

Für ein Objekt myTest der Klasse Test ist folgende Anweisung zulässig:

* + ~~myTest.iCount = 7;~~
  + ~~Test.iCount = 7;~~
  + myTest.setCount(7);

1. Der Name eines Konstruktors ist identisch mit dem Namen \_\_\_der Klasse\_\_\_\_\_\_\_\_.
2. Der Return-Typ eines Konstruktor
   * existiert nicht.
   * ~~ist mit dem Typ der Klasse identisch.~~
   * ~~ist void.~~
3. Konstruktoren derselben Klasse müssen unterscheidbar sein durch ihre \_\_Parameter\_\_\_\_.

# UML Pizzabäckerei

Modellieren Sie eine Pizzabäckerei, zu der folgende Informationen vorliegen:

* Eine Pizzabäckerei hat mindestens einen Backofen und
* mindestens einen Pizzabäcker.
* Sie benötigt ein Lager, das u.a. die Zutaten für Pizzen enthält.

In der Bäckerei können sich Kunden aufhalten, die auf die bestellten Pizzen warten. (Wir unterstellen hier, dass die Kunden in der Reihenfolge ihrer Ankunft bedient werden. Wenn ein Auftrag jedoch sehr lange dauert, kann es natürlich sein, dass Kunden mit bescheideneren Wünschen eher abgefertigt werden, obwohl sie später gekommen sind.)

Erstellen Sie nun ein Klassenmodell mit den wesentlichen Klassen und beschreiben Sie jeweils die Art der Beziehung, die Sie zwischen diesen Klassen ermitteln.



# Sparschwein

Implementieren Sie eine Klasse PiggyBank zur Verwaltung der Münzen in einem Sparschwein. Die Klasse besitzt folgende Attribute:

* Zähler für vier Arten von Geldstücken (Anzahl-1-Cent-, -10-Cent-, -50-Cent- und -1-Euro-Stücke)
* maximale Anzahl Geldstücke, die in das Sparschwein passen
* Flag, um anzuzeigen, dass das Sparschwein aufgebrochen wurde

Darüber hinaus deklariert die Klasse folgende Methoden:

|  |  |
| --- | --- |
| init() | Initialisiert jedes Attribut zur Darstellung eines leeren Sparschweins. Die Methode erwartet als Argument die maximale Anzahl von Münzen, die in das Sparschwein passen. |
| add1Cents() | "wirft" eine übergebene Anzahl von 1-Cent-Münzen in das Sparschwein und liefert den Return-Wert 0, falls alle Münzen in das Sparschwein passen. Wann das Sparschwein "überläuft", liefert die Methode die Anzahl Münzen, die nicht mehr hinein passen. |
| add10Cents()  add50Cents()  add1Euros() | analog zur Methode add1Cents() für den jeweiligen Münz-Typ |
| isEmpty() | liefert true, falls das Sparschwein leer ist, sonst false. |
| isFull() | liefert true, falls das Sparschwein voll ist, sonst false. |
| isBroken() | liefert true, falls das Sparschwein aufgebrochen ist, sonst false. |
| break() | bricht das Sparschwein auf. Der Zähler der Geldstücke wird auf 0 zurückgesetzt. Der Return-Wert ist der angesparte Geldbetrag in Cents. |

# Sparschwein testen

Testen Sie die Klasse PiggyBank mit einem Programm in einer separaten Quelldatei PiggyBankTest. Legen Sie ein Objekt der Klasse an, welches bis zu 500 Münzen speichern kann. "Werfen" Sie verschiedene Münzen in das Sparschwein, bis es voll ist. Brechen Sie dann das Sparschwein auf und zeigen Sie den gesparten Betrag an.

# Personalarray

Implementieren Sie eine Klasse “Mitarbeiter“. Mit den Attributen Vorname, Name, Personalnummer, Posten, Eintrittsdatum und Gehalt. Implementieren Sie entsprechende Konstruktoren und weitere notwendige Methoden.

Schreiben Sie ein Programm mit einem Array, in welchem 50 Mitarbeiter gespeichert werden können. Implementieren Sie Methoden, mit denen Mitarbeiter aufgrund der Personalnummer oder des Nachnamens aus dem Array ausgewählt und ausgegeben werden können.

Beispielinteraktion:

Eingabe: <Personalnummer>

Ausgabe: <Vorname>, <Name>, <Posten>, Firmenzugehörig seit <Datum> und <Gehalt>

# Lehrgangsobjekte

Implementieren Sie eine Klasse “Lehrgang“. Überlegen Sie sich, welche Attribute (außer Nummer, Name, Dauer) für diese Klasse sinnvoll sind.

Erzeugen Sie, in einem Programm, zwei Objekte dieser Klasse, die zwei der Lehrgänge repräsentieren, die Sie besuchen bzw. in der letzten Zeit besucht haben. Ihr Programm soll die Bezeichnung und die Nummer der Lehrgänge ausgeben.

# Topfspiel

Wir haben nun die Aufgabe ein kleines Topfspiel, mit objektorientiertem Ansatz, zu programmieren. Es ist ein Konsolenspiel zu realisieren, in dem die beiden Spieler an einem Rechner spielen. Die Eingaben erfolgen über Tastatur und die Ausgaben werden im Konsolenfenster dargestellt. Folgende Anforderungen soll das Spiel erfüllen:

* Das Spiel wird mit zwei Spielern gespielt.
* Jedem Spieler gehört ein kleiner, am Anfang leerer Topf.
* Nach Aufforderung des aktuellen Spielers (Spieler 1 oder Spieler 2) führt dieser einen möglichen Spielzug aus.
* Es wird abwechselt ein Spielzug von den Spielern durchgeführt. Spieler 1 beginnt das Spiel.
* Folgende 3 Aktionen sind möglich:
  + 1 – 3 Münzen in den eigenen Topf ablegen
  + Eine Münze aus dem gegnerischen Topf herausholen
  + Tauschen der Töpfe.
* Die möglichen Aktionen sind vor jedem Spielzug dem aktuellen Spieler anzuzeigen.
* Die ausgewählte Aktion gibt der Benutzer über die Tastatur dem Programm bekannt.
* Wenn die Aktion: Topftauschen ausgewählt wurde, dann ist diese Aktion für zwei weitere Züge nicht mehr erlaubt. Es ist eine Sperre zu integrieren.
* Nach einem Spielzug ist der aktuelle Spielstand auszugeben.
* Fehlerhafte Eingaben sind abzufangen und mit einer Meldung zu bestätigen.
* Gewonnen hat der Spieler, der zuerst 7 Münzen in seinem Topf hat.
* Eine Gewinnermeldung ist darzustellen und das Programm beendet sich.

1. Modellieren Sie die zwei Klassen:

* Eine Topf-Klasse: Topf, die die Anzahl der Münzen im Topf verwaltet und Methoden für das Hineinlegen oder Entfernen von Münzen anbietet, den aktuellen Topfinhalt ausgibt sowie Auskunft gibt, ob der Topf voll ist.
* Eine Spiel-Klasse: TopfSpiel, die den aktuellen Spieler, die Töpfe der Spieler und die Spielsperre verwaltet. Die benötigten Methoden entnehmen Sie dem vorgegebenen Hauptprogramm:

1. package topfSpiel;
2. public class TopfSpiel {
3. public static void main( String[] args ) {
4. TopfSpiel spiel = new TopfSpiel();
5. while ( !spiel.istBeendet() ) {
6. spiel.runde();
7. }
8. System.out.println( "Sieger ist Spieler " +   
    spiel.getSieger() );
9. }
10. }
11. Implementieren Sie das Spiel.

Vererbung und Polymorphie

# Verständnisfragen

Zunächst wieder ein paar Verständnisfragen vorne weg:

1. Eine "Hat-Beziehung" entsteht zwischen zwei verschiedenen Klassen, wenn
   * ~~eine Klasse von der anderen Klasse abgeleitet ist 🡪 ist Teil von~~
   * ein Attribut einer Klasse vom Typ der anderen Klasse ist
   * ~~eine Klasse innerhalb der anderen Klasse definiert ist~~
2. Eine in einer abgeleiteten Klasse überschriebene Methode kann die entsprechende public-Methode der Basisklasse aufrufen, und zwar mit Hilfe des Operators super()
3. Welche Beziehung besteht jeweils zwischen den Klassen

a) Person – Pkw – Person hat 1..n

b) Soldat – PK – Soldat hat 1 PK

c) KpChef – KpFw – KpChef hat 0..1 KpFw (Beide Richtungen)

d) Kp – KpFw – Kp hat 0..1 KpFw (Beide Richtungen)

e) Firma – Angestellter – Firma hat 0..n Angestellte

(Beide Richtungen je nach Formulierung

f) Firma – Kunde – Firma hat 0..n Kunden / Kunde hat 0..n Firmen die er

besucht

g) Tier – Affe – Ein Affe IST ein Tier

h) Zoo – Affenhaus – Der Zoo hat 0..n Affenhäuser

1. Was ist ein Konstruktor?

eine Methode, die den gleichen Namen wie die Klasse hat. Sie wird dazu verwendet, ein neues Objekt dieser einen Klasse anhand der entsprechenden Parameter zu erstellen.

1. Was sind die wesentlichen Konzepte der objektorientierten Programmierung?

[**1: Abstraktion** – Unterscheidung zwischen Objekten und Klassen](https://www.programmierenlernenhq.de/grundlagen-der-objektorientierten-programmierung-in-java/#Abstraktion)

**2: Kapselung** - Diese Zusammenfassung von Methoden und Variablen zu Klassen bezeichnet man als Kapselung. Sie stellt die zweite wichtige Eigenschaft objektorientierter Programmiersprachen dar. Kapselung hilft vor allem, die Komplexität der Bedienung eines Objekts zu reduzieren. Um eine Lampe anzuschalten, muß man nicht viel über den inneren Aufbau des Lichtschalters wissen. Sie vermindert aber auch die Komplexität der Implementierung, denn undefinierte Interaktionen mit anderen Bestandteilen des Programms werden verhindert oder reduziert.

[**3: Wiederverwendbarkeit** – Programmelemente mehrfach nutzen](https://www.programmierenlernenhq.de/grundlagen-der-objektorientierten-programmierung-in-java/#Wiederverwendbarkeit) (Methoden)

[**4: Beziehungen** – Objekte und Klassen stehen in Beziehungen zueinander](https://www.programmierenlernenhq.de/grundlagen-der-objektorientierten-programmierung-in-java/#Beziehungen)

[**5: Polymorphismus** – Polymorphie (gr. Vielgestaltigkeit) bezeichnet eine Sprachstruktur, nach der beim Zugriff auf Methoden mit identischer Signatur diese unterschiedlichen Ergebnisse liefern. Das Verhalten ist insbesondere im Zusammenhang mit Vererbung anzutreffen.](https://www.programmierenlernenhq.de/grundlagen-der-objektorientierten-programmierung-in-java/#Polymorphismus)

Statische Polymorphie

zeigt sich in Form des Überladens von Methoden, bei dem sich mehrere Methodensignaturen nicht in ihrem Bezeichner, jedoch in deren Parameterliste unterscheiden. Die Wahl der auszuführenden Methode geschieht somit anhand von Anzahl und Typ der Methodenparameter.

Bei der dynamischen Polymorphie

werden in mehreren Klassen einer Vererbungslinie gleiche, d.h. mit identischen Signaturen versehene Methoden unterschiedlich implementiert, sodass zur Laufzeit entschieden werden muss, welche der Methoden ausgeführt wird. Dies ist dann der Fall, wenn die Kindklassen die Methoden der Elternklasse überschreiben (nicht überladen!). Es muss dann zur Laufzeit entscheiden werden, ob die Methode der Eltern- oder diejenige der Kindklasse ausgeführt wird.