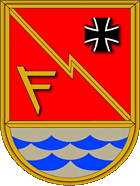
Schule Informationstechnik

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; /\* … \*/};

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

1. Elemente von verschiedenen Klassen
   * ~~müssen unterschiedliche Namen haben~~.
   * 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.

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~~
   * 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 \_\_\_\_\_.
3. Welche Beziehung besteht jeweils zwischen den Klassen

a) Person – Pkw

b) Soldat – PK

c) KpChef – KpFw

d) Kp – KpFw

e) Firma – Angestellter

f) Firma – Kunde

g) Tier – Affe

h) Zoo – Affenhaus

1. Was ist ein Konstruktor?
2. Was sind die wesentlichen Konzepte der objektorientierten Programmierung?

# Dreidimensionale Punkte

1. public class Punkt2D {
2. /\*
3. \* Koordinaten
4. \*/
5. public int x, y;
6. public Punkt2D (int x, int y) {
7. this.x = x;
8. this.y = y;
9. }
11. public boolean equals ( Object obj) {
12. boolean erg=false;
13. if(obj instanceof Punkt2D){
14. Punkt2D other=(Punkt2D) obj;
15. erg= (other.x == x) && (other.y == y) ;
16. }
17. return erg ;
18. }
19. public static void main(String [] args){
20. Punkt2D p1=new Punkt2D(2,2);
21. Punkt2D p2=new Punkt2D(3,2);
22. System.*out*.println(p1.equals(p2));
23. }
24. }

Die folgende Java-Klasse modelliert zweidimensionale Punkte:

Definieren Sie im gleichen Stil zur Klasse Punkt2D eine Unterklasse Punkt3D, deren Objekte dreidimensionale Punkte modellieren und daher zusätzlich eine z-Koordinate enthalten. Die Klasse Punkt3D soll folgende Elemente bereitstellen. Wo immer es geht, sollen sich diese Elemente auf Elemente der Oberklasse oder auf andere Elemente der eigenen Klasse abstützen:

1. Einen Konstruktor mit drei int-Parametern, mit denen die Koordinaten belegt werden.
2. Einen parameterlosen Konstruktor, der ein Exemplar des Ursprungspunkts (0,0,0) erzeugt.
3. Eine equals-Methode zum Vergleich von Punkt3D-Objekten mit anderen Punkt3D-Objekten.
4. Ergänzen Sie die beiden Klassen um eine sinnvolle toString() Methode.

# **Kreis im geo-Paket**

Wir haben gestern begonnen das geo-Paket zu implementieren. Nehmen Sie folgende Ergänzungen vor:

Ergänzen Sie die Klasse Kreis. Ordnen Sie diese entsprechend im Paket ein. Sie soll mindestens die folgenden Funktionen haben:

* 1. Einen Konstruktor, der einen Radius und den Mittelpunkt bekommt
  2. bewege(int x, int y): Verschiebt den Kreis um den Wert x horizontal und y vertikal.
  3. abstand(Punkt punkt): Berechnet den Abstand zwischen einen Punkt und dem Kreis (Gehen Sie objektorientiert vor. Vielleicht kann ja eine Abstandsfunktion in der Klasse Punkt helfen).
  4. toString(): Gibt den Kreis geeignet als String aus.

# Fahrzeuge

Es sollen Fahrzeuge zunächst modelliert werden. Vorausgesetzt sei folgende Aufteilung eines Pakets *fahrzeuge* in Klassen:

1. Erstellen Sie die Interfaces *Fahrzeug*, *Auto*, *Schiff* und *Amphibienfahrzeug*.
2. Erstellen Sie die Klasse *AutoImpl*. Implementieren Sie neben dem Konstruktor AutoImpl(double gewicht), der *gewicht* setzt, zwei Methoden double getGewicht(), die das Gewicht zurückliefert, und void closeLock(boolean b), die die Zentralverriegelung steuert.
3. Erstellen Sie die Klasse *SchiffImpl*. Implementieren Sie neben dem Konstruktor SchiffImpl(double gewicht), der *gewicht* setzt, zwei Methoden double getGewicht(), die das Gewicht zurückliefert, und void closeGangway(boolean b), die die Gangway steuert.
4. Erstellen Sie die Klasse *AmphibienfahrzeugImpl*. Implementieren Sie neben dem Konstruktor AmphibienfahrzeugImpl(double gewicht) weitere evtl. notwendige Methoden. void closeGangway(boolean b) gibt die Ausführung an die entsprechende Methode von *schiffImpl* weiter.
5. Schreiben Sie in der main-Funktion Anweisungen, die ein Objekt der Klasse AmphibienfahrzeugImpl mit Gewicht 3000.0 erzeugt, die Gangway einfährt und das Fahrzeug versperrt.

