

Aufgaben: Objekte, Klassen & UML

Bearbeite die folgenden Aufgaben in Java. Die Schwierigkeit nimmt von Aufgabe zu Aufgabe nicht unbedingt linear zu - lies die Aufgabenstellung genau, bevor du anfaengst zu programmieren. Bei Aufgaben mit UML-Diagramm: Setze JEDES Detail (Attribute, Sichtbarkeiten, Methodensignaturen) korrekt um.

Aufgabe 1 — Klassen & Kapselung

Schwierigkeit: mittel

```
/** Erstelle eine Klasse 'Mitarbeiter' mit den privaten Attributen
    name (String), geburtsjahr (int) und gehalt (double).
    Schreibe ZWEI Konstruktoren: einen mit allen drei Parametern und
    einen mit nur name und geburtsjahr (gehalt wird dann auf 2000.0
    gesetzt).
    Das Gehalt darf NIE negativ sein - verwende dafuer eine Setter-
    Methode mit Validierung; bei einem ungueltigen Wert bleibt das
    alte Gehalt erhalten und es wird eine Fehlermeldung ausgegeben.
    Schreibe ausserdem eine Methode 'gibBonus(double prozent)', die
    das Gehalt um den angegebenen Prozentsatz erhoeht.
    Ueberschreibe die Methode toString(), sodass z.B.
    "Anna Mueller (geb. 1990): 2500.0 Euro" ausgegeben wird.
    Zaehle ausserdem mit einer STATISCHEN Variable, wie viele
    Mitarbeiter-Objekte insgesamt erzeugt wurden, und gib diese Zahl
    am Ende in main() aus.
*/

class Mitarbeiter {
    // TODO: Attribute, Konstruktoren und Methoden ergaenzen
}

class Firma {
    public static void main( String[] args ) {
        // TODO: Erzeuge mindestens 3 Mitarbeiter-Objekte (nutze BEIDE
        // Konstruktoren mindestens einmal), gib jedem einen Bonus von
        // 5%, und drucke alle Mitarbeiter sowie die Gesamtanzahl der
        // erzeugten Objekte aus.
    }
}
```

Aufgabe 2 — Objekte & Arrays von Objekten

Schwierigkeit: mittel-schwer

```
/** Fuelle zunaechst ein Array 'lager' der Laenge 8 mit Objekten der
    Klasse 'Produkt'. Jedes Produkt besitzt einen Namen (aus dem
    Array 'namen' uebernehmen), einen Preis zwischen 5.0 und 200.0
    sowie eine Lagermenge (int) zwischen 1 und 50 (beides zufaellig).
    Berechne und gib den GESAMTWERT des Lagers aus (Summe aus
    Preis * Menge fuer jedes Produkt).
    Sortiere das Array anschliessend NACH PREIS aufsteigend -
    implementiere dafuer SELBST einen Bubble-Sort (KEINE
    Arrays.sort() oder Comparator verwenden!) und gib das sortierte
    Array aus.
    Gib am Ende zusaetzlich aus, welches Produkt den hoechsten
    Lagerwert (Preis * Menge) besitzt.
*/
import java.util.Random;      // random number generator

class Produkt {
    // TODO: Attribute (name, preis, menge), Konstruktor, toString()
}

class Arrays03 {
    public static void main( String[] args ) {
        Random rand = new Random();
        String[] namen = { "Tastatur", "Monitor", "Maus", "Drucker",
                           "Kabel", "Lautsprecher", "Webcam", "Headset" };
        // Tipp: rand.nextDouble() liefert eine Zahl zwischen 0.0 und 1.0
        // Tipp: rand.nextInt( 50 ) liefert eine Zahl zwischen 0 und 49
    }
}
```

Aufgabe 3 — Vererbung & Polymorphismus (UML -> Code)

Schwierigkeit: schwer

Im folgenden UML-Diagramm ist eine abstrakte Klasse **Fahrzeug** mit zwei Unterklassen dargestellt. Setze dieses Diagramm 1:1 in Java-Code um.

```

+-----+
| <<abstract>> Fahrzeug |
+-----+
| - marke : String      |
| - baujahr : int       |
| - geschwindigkeit : double |
+-----+
| + Fahrzeug(marke, baujahr) |
| + beschleunigen(delta:double) |
| + bremsen(delta:double) |
| + toString() : String |
| + gibTyp() : String {abstract}|
+-----+
      /_ \
      |
+-----+-----+
| | | | | | | |
+-----+-----+
| Auto | | Motorrad |
+-----+-----+
| -anzahlTueren | | -hatBeiwagen |
| : int | | : boolean |
+-----+-----+
| +Auto(...) | | +Motorrad(...) |
| +gibTyp() | | +gibTyp() |
+-----+-----+

```

- geschwindigkeit startet bei 0 und darf durch beschleunigen()/bremsen() NIE negativ werden; begrenze sie ausserdem sinnvoll nach oben (z.B. 250 km/h) - die Grenze gehoert zu Fahrzeug.
- gibTyp() ist abstrakt in Fahrzeug, liefert in Auto "Auto" und in Motorrad "Motorrad".
- Erzeuge im main ein Array vom Typ Fahrzeug[] der Laenge 5 und fuehle es GEMISCHT mit Auto- und Motorrad-Objekten.
- Iteriere POLYMORPH (also ueber die Fahrzeug-Referenz!) ueber das Array, lasse jedes Fahrzeug beschleunigen und gib es aus.
- Zaehle anschliessend mit instanceof, wie viele Autos und wie viele Motorraeder im Array enthalten sind.

```

abstract class Fahrzeug {
    // TODO

}

class Auto extends Fahrzeug {
    // TODO

}

class Motorrad extends Fahrzeug {
    // TODO

}

class FahrzeugTest {
    public static void main( String[] args ) {
        // TODO

    }
}

```

Aufgabe 4 — Aggregation & Code -> UML

Schwierigkeit: mittel-schwer

```
/** Eine 'Bibliothek' verwaltet eine beliebige Anzahl von 'Buch'-
    Objekten (Aggregation: ein Buch kann auch OHNE Bibliothek
    existieren). Jedes Buch besitzt einen Titel, einen Autor und
    einen Status 'ausgeliehen' (boolean, initial false).

    Ergaenze die Klasse Bibliothek so, dass:
    - Buecher ueber addBuch(Buch b) hinzugefuegt werden koennen
      (ArrayList verwenden)
    - leiheAus(String titel) das erste passende, NICHT bereits
      ausgeliehene Buch ausleiht (Status setzen) und true
      zurueckgibt, sonst false
    - gibZurueck(String titel) den Status wieder zuruecksetzt
    - zeigeVerfuegbare() alle aktuell NICHT ausgeliehenen Buecher
      ausgibt

    ZUSATZAUFGABE (schriftlich, NICHT im Code):
    Zeichne fuer Bibliothek und Buch ein UML-Klassendiagramm.
    Achte dabei besonders auf:
    - die richtige Beziehungsart (Aggregation oder Komposition?)
    - die Multiplizitaet auf BEIDEN Seiten der Beziehung
    - die Sichtbarkeiten (+/-) der Attribute und Methoden
*/
import java.util.ArrayList;

class Buch {
    // TODO
}

class Bibliothek {
    // TODO

    public static void main( String[] args ) {
        // TODO: Erzeuge eine Bibliothek, fuege 4 Buecher hinzu, leihe
        // 2 davon aus, gib eines zurueck und zeige am Ende die
        // verfuegbaren Buecher an.
    }
}
```

Aufgabe 5 — Statische Attribute & Methoden

Schwierigkeit: mittel

```
/** Erstelle eine Klasse 'Konto' mit den privaten Attributen
    kontonummer (int), inhaber (String) und kontostand (double).
    Die Kontonummer wird NICHT vom Benutzer vergeben, sondern
    automatisch hochgezählt (erstes Konto = 1000, danach jeweils
    +1) - verwende dafür eine STATISCHE Variable.
    Führe ausserdem eine STATISCHE Variable 'gesamtVermoeegenBank',
    die bei jeder Einzahlung/Abhebung automatisch aktualisiert wird
    (Summe aller Kontostaende ueber ALLE Konten).

    Methoden:
    - einzahlen(double betrag): erhoeht Kontostand und
      gesamtVermoeegenBank; bei negativem Betrag: Fehlermeldung,
      nichts passiert
    - abheben(double betrag): wie einzahlen, aber Abhebung nur
      erlaubt, wenn genug Geld auf dem Konto ist, sonst
      Fehlermeldung
    - toString(): z.B. "Konto 1000 (Hans Mueller): 250.0 Euro"

    Schreibe zusaetzlich eine STATISCHE Methode
    gibGesamtvermoeegenDerBank(), die den aktuellen Wert von
    gesamtVermoeegenBank zurueckgibt.
*/

class Konto {
    // TODO
}

class Bank {
    public static void main( String[] args ) {
        // TODO: Erzeuge 4 Konten, fuehre verschiedene Ein- und
        // Abhebungen durch (auch MINDESTENS einen Versuch, der
        // fehlschlagen MUSS), und gib am Ende alle Konten sowie das
        // Gesamtvermoeegen der Bank aus.
    }
}
```